TAMPEREEN TEKNILLINEN YLIOPISTO

PETRI ÖSTERMAN

UNIQ-ALUSTAN TIEDONSIIRTOKYVYN MITTAUS JA OPTIMOINTI

Diplomityö

Tarkastaja: professori Hannu Koivisto
Tarkastaja ja aihe hyväksytty
Teknisten tieteiden tiedekuntaneuvoston kokouksessa
3. kesäkuuta 2015

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
ÖSTERMAN, PETRI: UniQ-alustan tiedonsiirtokyvyn mittaus ja optimointi
Diplomityö, 57 sivua
Helmikuu 2017
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Hannu Koivisto
Avainsanat: Tiedonsiirto, optimointi, suorituskyky

Tiedon tehokas siirtäminen internetissä ja muissa tietoverkoissa on haastava prosessi, jota on tutkittu vuosikymmeniä. Tehokkaalla tiedonsiirrolla tarkoitetaan suurta tiedonsiirtonopeutta, pientä viivettä tiedon siirtymisessä sekä varmuutta tiedon välittymisestä vastaanottajalle.

UniQ-alusta on kehitetty satama-automaation tarpeita varten. Sen keskeisimpiä tehtäviä on tarjota käyttäjille erinäisiä palveluita ja varmistaa tiedon siirtyminen vastaanottajalle. Tieto pitäisi siirtää lisäksi suurella nopeudella ja pienellä viiveellä.

Tämän diplomityön tavoitteena on mitata UniQ-alustan suorituskykyä ja pyrkiä parantamaan sitä. Suorituskyky käsittää tiedonsiirtokyvyn, tarvittavan laskentatehon sekä viestien välittämiseen kuluvan ajan.

Työtä varten pystytettiin tarkoitukseen sopiva mittausympäristö sekä toteutettiin erillinen mittausohjelma ja automaattitestit, joilla pystyttiin mittaamaan edellä mainittuja suorituskykyarvoja. Tulokset dokumentoitiin myöhempää vertailua varten, minkä jälkeen ryhdyttiin selvittämään alustan suorituskykyongelmia analysointityökalujen ja koodikatselmoinnin avulla. Kaikkien toteutettujen optimointien vaikutus todennettiin automaattitesteillä. Merkittävät optimoinnit dokumentoitiin ja ne on käyty läpi tässä raportissa.

Työn tulokset ovat merkittävät kaikkien mitattujen suorituskykyarvojen osalta. Tiedonsiirtonopeus parani monikymmenkertaiseksi ja alustan prosessorin käyttö aleni murto-osaan alkuperäisestä. Lisäksi tiedonvälitykseen kuluva aika pieneni prosentuaalisesti merkittävästi.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
ÖSTERMAN, PETRI: Measuring and optimizing the performance of The UniQ Platform
Master of Science Thesis, 57 pages
February 2017
Major: Automation software engineering
Examiner: Professor Hannu Koivisto
Keywords: Data transfer, optimizing, performance

Efficient data transfer in Internet and other networks is a challenging process which has been studied for decades. The efficient data transfer consists of high data transfer rate, small transfer delay and secure data delivery.

The UniQ Platform has been developed for port automation. Its main goal is to offer different services and deliver data securely. In addition, data should be transferred with high rate and small delay.

Purpose of this thesis is to measure and to optimize the performance of the UniQ Platform. The Performance consists of data transfer rate, processor usage and transfer delay.

An appropriate environment was constructed to measure the performance of the platform. Actual measuring was done with aid of automated tests and program developed particularly for the purpose. Results were documented and performance problems were examined with analyzing tools and code reviewing. Benefit of all implemented performance optimizations were measured with automated tests. Major optimizations were documented and explained in this thesis.

The Results were significant. Data transfer rate was improved 30 – 200 times with regular packet sizes and processor usage was reduced to a fraction of the original usage. Transfer delays were also reduced significantly.

ALKUSANAT

Tämä opinnäytetyö on Cargotec OY:lle tehty diplomityö. Työ on suoritettu Tampereen yksikölle muiden työtehtävien ohella. Diplomityön tarkastajana ja ohjaajana toimi professori Hannu Koivisto Tampereen Teknillisestä Yliopistosta.

Haluan kiittää Cargotec OY:tä saamastani mahdollisuudesta toteuttaa diplomityöni yritykselle. Kiitoksen ansaitsevat myös Tane Reinikainen ja Aleksi Lehtonen hyvästä perehdytyksestä sekä työn ohjaajana toiminut professori Hannu Koivisto kärsivällisestä ja motivoivasta asenteesta.

Erityiskiitoksen ansaitsee kihlattuni. Hänen tukensa vaikutti merkittävästi siihen, että sain vietyä työn päätökseen.

Tampereella 20.1.2017

Petri Österman

SISÄLLYS

TERMIT JA LYHENTEET

Asiakaskirjasto	Spread Toolkitin asiakaskirjasto.
Ikkunankasvatus-funktio	TCP:n lähetysikkunan koon määräävä funktio. (window growth function)
Kirjoitusloki	Tietokantojen käyttämä väliaikaistiedosto, joka sisältää tietokannan viimeisimmät muutokset. (write-ahead log, WAL)
Kuittausviive	Viive, joka kuluu viestin lähettämisestä siihen, että sen perille menosta saadaan kuittaus. (round-trip time, RTT)
Levyvarmennettu viesti	Viesti, joka puskuroidaan sellaiseen muistiin, joka säilyy virtakatkojen yli (esim. kiintolevy). Viesti uudelleenlähetetään pyydettäessä.
Muistivarmennettu viesti	Viesti, joka puskuroidaan työmuistiin ja uudelleenlähetetään pyydettäessä.
Palautusmuistio	Tietokantojen käyttämä varmuuskopio. Kopiota hyödynnetään, jos muutosten tekeminen kantaan keskeytetään ja halutaan palata alkuperäiseen tilaan. (rollback journal)
Palvelinohjelma	Spread Toolkitin keskuspalvelin.
PPnS	Tiedonsiirtoalustan kerros, jonka tehtävänä on huolehtia yhteyksien ylläpitämisestä sekä viestien puskuroinnista. (Persistent publish and subscribe)
Ruuhkanhallinta	Verkon liiallisen kuormittumisen estäminen. (congestion control)
SQLite	Kevyt versio SQL-tietokannasta.
Sykeviesti	Viesti, jolla varmistetaan yhteyden toimivuus. Jos viestejä ei saada riittävän pitkään aikaa, yhteyden tiedetään katkennen. (heartbeat-message)
Tag Facade	Tiedonsiirtoalustan kerros, jonka tehtävänä on tarjota erinäisiä palveluita ylemmän tason kerrosten käyttöön.
TCP	Protokolla, joka huolehtii tiedon luotettavasta siirtämisestä verkon kahden koneen välillä. (transport control protocol)
TCP Cubic	TCP:n muunnos, jossa on suurinopeuksisiin ja -viiveisiin verkkoihin soveltuva ruuhkanhallintaikkuna-algoritmi.
TCP Westwood	TCP:n muunnos, jossa on langattomiin verkkoihin suunniteltu ruuhkanhallintaikkuna-algoritmi.

Varmentamaton viesti	Viesti, joka puskuroidaan muistiin lähetystä varten, mutta sitä ei säilötä uudelleenlähetystä varten.
Vuonohjaus	Lähetysnopeuden kontrollointi. (flow control)
Vuorontaja	Ohjelman osa, joka valitsee lähetettävän viestin. (scheduler)
Välitön viesti	Viesti, jota ei puskuroida muistiin vaan lähetetään välittömästi.
WANem	Laajojen verkkojen emulointiin tarkoitettu työkalu. (Wide Area Network Emulator)

1 JOHDANTO

Tässä diplomityössä on pyritty ratkaisemaan kaksi ongelmaa, jotka nivoutuvat hyvin yhteen. Ensinnäkin asiakasta on kiinnostanut tietää oman tiedonsiirtoalustansa suorituskyky. Tässä tapauksessa suorituskyky käsittää tiedonsiirtokapasiteetin (eng. throughput), prosessorikuorman ja viestin välittämiseen kuluvan viiveen.

Varsinaisena ongelmana on ollut alustan raskaus ja riittämätön suorituskyky, joten tavoitteena on ollut myös tiedonsiirtokapasiteetin kasvattaminen ja prosessorikuorman sekä viestien välitykseen kuluvan ajan (viiveen) pienentäminen.

Ongelmia on lähdetty ratkaisemaan mainitussa järjestyksessä. Automatisoidut suorituskykytestit ratkaisevat ensimmäisen ongelman ja toimivat välttämättöminä työkaluina suorituskyvyn parantamisessa. Niiden avulla on todennettu toteutettujen optimointien vaikutus suorituskykyyn.

Tiedonsiirtoalustan tyypillinen käyttöympäristö on satama, jossa lastataan ja puretaan kontteja rahtilaivoista. Satamassa voi olla sekä manuaalisesti että automaattisesti toimivia nostureita ja muita laitteita, joilla siirrellään kontteja paikasta toiseen.

Satama-alueelle on sijoitettu useita WLAN-tukiasemia, jotta verkon peittoalue olisi mahdollisimman kattava. Siitä huolimatta konttien väleihin jää helposti alueita, joissa kuuluvuus on heikko tai sitä ei ole lainkaan. Tästä johtuen konttien väleissä liikkuvat muutaman metrin korkuisten kontinsiirtolaitteiden yhteys katkeilee vähän väliä. Suurten, jopa kerrostalon korkuisten nostureiden kohdalla yhteys on paljon vakaampi, sillä niiden antennit sijaitsevat selvästi konttipinojen yläpuolella. Siitä huolimatta langaton yhteys on aina häiriöaltis.

Käsiteltävä tiedonsiirtoalusta on monitasoinen. Siinä on tämän työn näkökulmasta neljä selvää kerrosta, joista jokainen asettaa omat rajoitteensa siirtokapasiteetille. Kerrokset on esitelty tarkemmin luvussa 2.

Optimoinnissa keskitytään pääasiassa PPnS-kerrokseen (persistent publish and subscribe), joka kuvataan tarkemmin omassa luvussaan (luku 3). PPnS-kerroksen tehtävänä on havaita yhteyksien katkeaminen nopeasti sekä huolehtia viestien puskuroinnista ja uudelleenlähetyksestä yhteyden katketessa.

Luvusssa 4 käydään läpi mittauksen tavoitteet ja käytetyt menetelmät, mittausympäristö sekä mittauksissa ja optimoinneissa käytetyt työkalut. Valmiita työkaluja hyödynnettiin mahdollisuuksien mukaan, mutta suorituskykymittauksien suorittamiseen toteutettiin tarkoitukseen räätälöity sovellus. Luvussa käydään lisäksi läpi tiedonsiirtoalustan suorituskyky työn aloitushetkellä.

Luvussa 5 käsitellään alustaan toteutetut optimoinnit. Osalla optimoinneista oli merkittävä vaikutus suorituskykyy ja joidenkin vain marginaalinen. Luvussa ei käydä läpi marginaalisia optimointeja.

Luvussa 6 käydään läpi alustan suorituskyky optimointien jälkeen. Saavutetu tulokset ovat merkittäviä kaikilla työssä tarkastelluilla osa-alueilla (tiedonsiirtokyky, prosessorikuorma ja viive). Tiedonsiirtokyky kasvoi monin paikoin jopa monikymmenkertaiseksi ja prosessorikuorma pieneni lähes samassa suhteessa. Tiedonsiirtoviiveen pienenemisessä tulokset olivat maltillisimpia, mutta nekin pienenivät merkittävästi prosentuaalisesti.

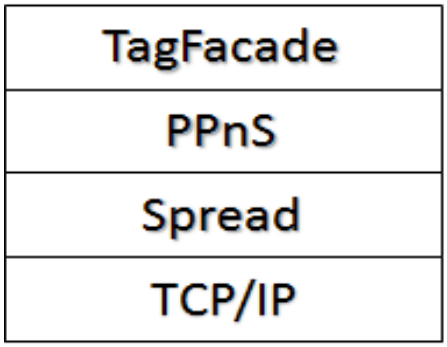
Luvussa 7 esitetään jatkokehitysideoita, jotka voidaan toteuttaa tulevaisuudessa. Luvussa 8 on lyhyt yhteenveto työn tuloksista.

2 LÄHTÖKOHDAT

Tässä luvussa pyritään antamaan yleiskuva tiedonsiirtoalustan rakenteesta käymällä läpi lyhyesti sen jokainen siirtokerros. Luvussa käydään läpi myös muuta työn kannalta keskeistä teoriaa.

2.1 Tiedonsiirtoalusta

Käsiteltävä tiedonsiirtoalusta voidaan jakaa neljään kerrokseen. Alimpana tarkasteltavana kerroksena toimii TCP (Transport Control Protocol). Tämän päällä toimii vapaan lähdekoodin Spread Toolkit, joka huolehtii viestien välityksestä laitteiden välillä. Spread Toolkit ei ole suunniteltu toimimaan huonossa langattomassa verkossa, jossa yhteydet katkeilevat vähän väliä. Sen puutteita paikataan PPnS-kerroksella, joka pyrkii havaitsemaan yhteyden katkeamiset nopeasti ja huolehtimaan nopeasta uudelleen yhdistämisestä. Neljäntenä kerroksena toimii Tag Facade, joka tarjoaa tiedonsiirtoalustaa käyttäville sovellusohjelmille erilaisia palveluita. Tyypillinen esimerkki tällaisesta palvelusta on jonkin arvon tai arvojen lähettäminen muutoksesta tai tietyin aikavälein. Valvomosta voidaan esimerkiksi pyytää jotakin laitetta päivittämään sijaintinsa tietyin aikavälein. Kuva 1 esittää tarkasteltavaa osaa tiedonsiirtoalustan protokollapinosta.



Kuva 1: Tarkasteltava osuus tiedonsiirtoalustan protokollapinosta.

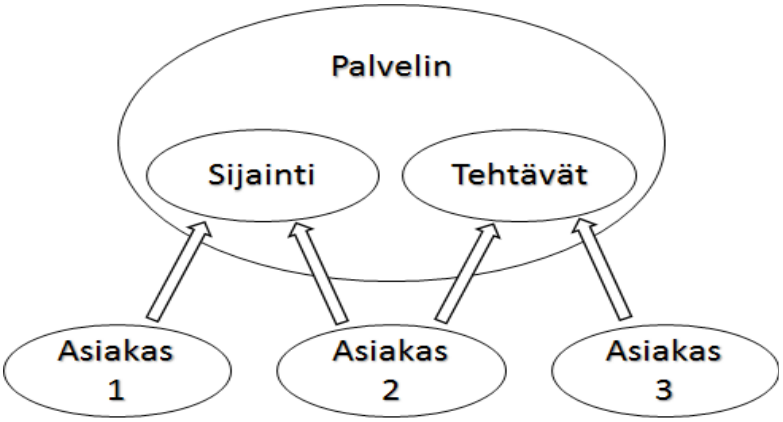
Näiden kerrosten alapuolella on siirtokerros (tyypillisesti WLAN tai LAN) sekä fyysinen kerros. Jokaisesta kuvassa 1 esitetystä kerroksesta on oma alalukunsa, jossa käydään läpi kerroksen tehtävät ja toiminta. Tarkoituksena on, että lukijalle muodostuu kuva kunkin kerroksen vastuualueista ja suhteestaan muihin kerroksiin. Työn kannalta keskeisin kerros on PPnS, joten se käydään tarkemmin läpi omassa luvussaan.

2.1.1 Spread Toolkit

Spread Toolkit on Spread Concepts LCC:n kehittämä vapaan lähdekoodin ohjelmisto, joka tarjoaa nopean ja luotettavan tavan välittää tietoa. Sen vahvoja ominaisuuksia ovat yksinkertainen, mutta tehokas rajapinta sekä viestien priorisointi [1].

Spread Toolkit toimii perinteisen asiakas-palvelin -mallin mukaisesti. Lisäksi se tukee usean palvelimen yhdistämistä keskenään, jolloin palvelimeen A yhdistänyt asiakas pystyy lähettämään viestin palvelimeen B yhdistäneelle asiakkaalle. Vaikka Spread Toolkit tukeekin tällaisia usean palvelimen verkkoja, niille ei ole käytännön tarvetta, joten tässä kappaleessa käsittellään yksinkertaisuuden vuoksi vain yhden palvelimen verkkoja.

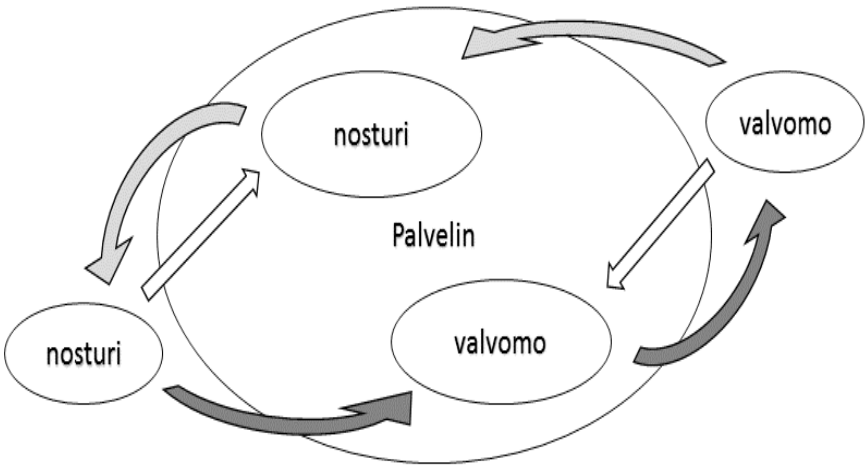
Spread Toolkit koostuu itsenäisestä palvelinohjelmasta (Spread daemon) sekä asiakaskirjastosta (Spread client). Asiakaskirjastoa käyttävä ohjelma yhdistää palvelimeen, jonka jälkeen se voi liittyä haluamiinsa ryhmiin. Ryhmiin voi lähettää viestejä ilman, että niihin on erikseen liittynyt. Viesti välitetään niille asiakkaille, jotka ovat liittyneet siihen ryhmään, johon viesti on lähetetty. Kuva 2 selventää tilannetta.



Kuva 2: Esimerkki Spread Toolkitin palvelimen, asiakkaiden ja ryhmien suhteesta.

Kuvassa 2 kaikki asiakkaat ovat yhdistäneet palvelimeen. Asiakkaat 1 ja 2 ovat liittyneet sijainti-ryhmään ja asiakkaat 2 ja 3 ovat liittyneet tehtävät-ryhmään. Kun asiakas 1 lähettää viestin sijainti-ryhmään, viestin saa hänen itsensä lisäksi asiakas 2. Jos asiakas 1 lähettää viestin tehtävät-ryhmään, viestin saavat vain asiakkaat 2 ja 3.

Tiedonsiirtoalusta ei hyödynnä Spread Toolkitin tarjoamaa ryhmäviestintää siinä laajuudessa, kuin se olisi mahdollista vaan ryhmät kuvaavat yksittäisiä laitteita. Kuva 3 on selventää asiaa.



Kuva 3: PPnS:n tapa käyttää Spread Toolkitia viestien välitykseen kahden laitteen välillä.

Kuvassa 3 on esitetty, kuinka PPnS-taso käyttää Spread-tasoa kommunikointiin. Jokainen verkkoon liittyvä laite yhdistää palvelimeen ja liittyy kuuntelemaan oman nimistänsä ryhmää. Esimerkissä nosturi-niminen laite kuuntelee nosturi-ryhmää ja valvomo valvomo-nimistä ryhmää. Kun valvomo haluaa lähettää viestin nosturille, se lähettää sen nosturi-ryhmään. Koska nosturi kuuntelee nosturi-nimistä ryhmää, viesti välitetään sille. Vastaavasti nosturin tulee lähettää valvomolle tarkoitetut viestit valvomo-ryhmään. Huomattavaa on, että jokaista ryhmää kuuntelee aina vain yksi laite. Idea on vastaava, kuin IRC-serverissä (Internet Relay Chat) [2].

2.1.2 PPnS

PPnS (Persistent publish and subscribe) taso on toteutettu kokonaisuudessaan asiakkaan toimesta käyttäen Qt:ta ja C++:aa. Se käyttää suoraan Spread Toolkitin asiakaskirjaston rajapintaa.

PPnS tasolla on kaksi keskeistä tehtävää. Tiedonsiirtoalustaa käytetään paljon langattomissa verkoissa, ja katvealueilla yhteydet voivat olla olemattomia. Tästä syystä yhteydet voivat katkeilla useinkin. Spread Toolkit ei selviydy yhteyksien katkeilemisista riittävän tehokkaasti, joten PPnS:n tehtävänä on pyrkiä havaitsemaan yhteyden katkeaminen nopeasti ja luoda yhteys uudelleen heti kun se on mahdollista. Käytännössä tämä toteutetaan siten, että laitteet lähettävät toisilleen sykeviestejä (eng. heartbeat message) ilmoittaen olemassaolostaan. Jos laite ei saa sykeviestiä riittävän usein, se tulkitsee vastapään yhteyden katkenneeksi ja pysäyttää informaatiota sisältävien viestien lähettämisen jatkaen kuitenkin itse sykeviestien lähettämistä. Kaikki laitteet ovat lisäksi liittyneet yhteiselle sykeviestikanavalle. Jos laite ei saa sykeviestiä riittävän usein tältä kanavalta, se lähettää sen sinne itse. Siinä tilanteessa, että laite ei saa olemassaoloviestiä ollenkaan sykeviestikanavalta, se tulkitsee oman yhteytensä

katkenneeksi, lopettaa kaikkien viestien lähettämisen ja pyrkii yhdistämään palvelimeen uudelleen.

PPnS:n toinen tehtävä on tarjota takuu viestien välittymisestä vastaanottajalle (eng. quality of service). Spread Toolkit tarjoaa eritasoisia takuita viestien välitykseen, mutta ne eivät ulotu yhteyden katkeamisen yli. PPnS tarjoaa neljä eritasoista takuuta viestien välitykseen. Nämä takuut säilyvät vaikka laitteiden välinen yhteys katkeaisikin. PPnS:n tarjoamat takuut viestien välityksestä on esitetty taulukossa 1.

Taulukko 1: PPnS:n tarjoamat takuut viestien välityksestä.

Viestin luokka	**Varmen-nettu**	**Järjestys-numero**	**Kuvaus**
Välitön viesti	Ei	Ei	Lähetetään välittömästä ohi muiden viestijonojen. Perille menosta ei taata mitään.
Varmentamaton	Ei	Kyllä	Viestit saapuvat määränpäähän samassa järjestyksessä, kuin ne lähetetään.
Muistivarmennettu	Kyllä	Kyllä	Viestien perillemeno taataan vaikka yhteys katkeaisi välillä.
Levyvarmennettu	Kyllä	Kyllä	Viestien perillemeno taataan vaikka yhteydet katkeisivat tai laitteesta katoaisi virta.

Välittömille viesteille ei anneta PPnS:n taholta minkäänlaista takuuta, joten niille pätevät vain Spread Toolkitin ja TCP:n antamat takuut. Takuiden puutteesta on se hyöty, että viesti voidaan lähettää heti, koska sitä ei tarvitse kierrättään minkään jonon kautta. Välitön viesti on nopein tapa välittää tieto, mutta sen perille menosta ei ole takuuta.

Varmentamattomattomien viestien perillemenoa ei myöskään taata, mutta ne lisätään jonoon, joten ne voidaan puskuroida yhteyden katkeamisen ajaksi. Puskuroidut viestit lähetetään, kun yhteys palautuu.

Muistivarmennettujen viestien taataan menevän perille siinä tapauksessa, että lähettävän laitteen virrat eivät katkea. Lisäksi taataan, että viestit käsitellään siinä järjestyksessä, kuin ne on käyttäjän toimesta lähetetty.

Levyvarmennetuille viesteille annetaan kaikkein korkein takuu perillemenosta. Viestien taataan menevän perille, vaikka lähettävästä laitteesta katkeaisi jostain syystä virrat. Tämä vaatii, että viestit sisältävän jonon täytyy sijaita sellaisessa muistissa, jonka sisältö säilyy virtakatkon yli. Tällaiset muistit ovat yleisesti selvästi työmuistia hitaampia, joka aiheuttaa selvän pullonkaulan suorituskyvylle.

Kunkin viestiluokan toteutus on kuvattu tarkemmin luvussa 3. Lisäksi PPnS sisältää muutamia sisäisiä viestiluokkia, joilla on omat prioriteetit ja takuut. Näitä luokkia ei käsitellä tässä työssä, koska niissä käytetään samoja mekanismeja, kuin edellä mainituissa luokissa.

2.1.3 Tag Facade

Tag Facade toimii PPnS:n päällä ja käyttää suoraan sen tarjoamia palveluita. Tag Facaden tehtävä on tarjota tiedonsiirtoalustaa käyttäville sovellusohjelmille erilaisia palveluita, jotka helpottavat monien ominaisuuksien toteuttamista. Tässä työssä ei keskitytä tarkemmin Tag Facaden suorituskyvyn optimointiin, mutta sen tyypilliset käyttötapaukset käydään läpi. Tämä siksi, että lukijalle muodostuu kuva siitä, millaisia palveluita PPnS-kerros välittää.

Kaikista eniten käytetty ominaisuus on tiedonlähetyspyyntö. Tiedonlähetyspyyntö lähetetään useimmiten valvomosta tai muusta vastaavasta paikasta, jossa halutaan monitoroida yhden tai useamman laitteen tietoja. Pyynnön saaja on tavallisesti jokin satamassa toimiva laite, esimerkiksi nosturi. Pyynnössä kerrotaan, mitä tietoja halutaan monitoroida ja kuinka usein tiedot lähetetään. Tällainen pyyntö lähetetään yleensä varmennettuna viestinä. Vastauksena saatava, jatkuvasti tai muutoksesta päivittyvä tieto välitetään varmentamattomana, sillä vanhoilla arvoilla ei yleensä ole mitään merkitystä. Lisäksi TCP ja Spread Toolkit pitävät yhdessä huolen siitä, että viestit välittyvät määränpäähän oikeassa järjestyksessä kunhan yhteydet eivät katkea. Varmentamattomat viestit voivat kadota vain katkenneiden yhteyksien aikana.

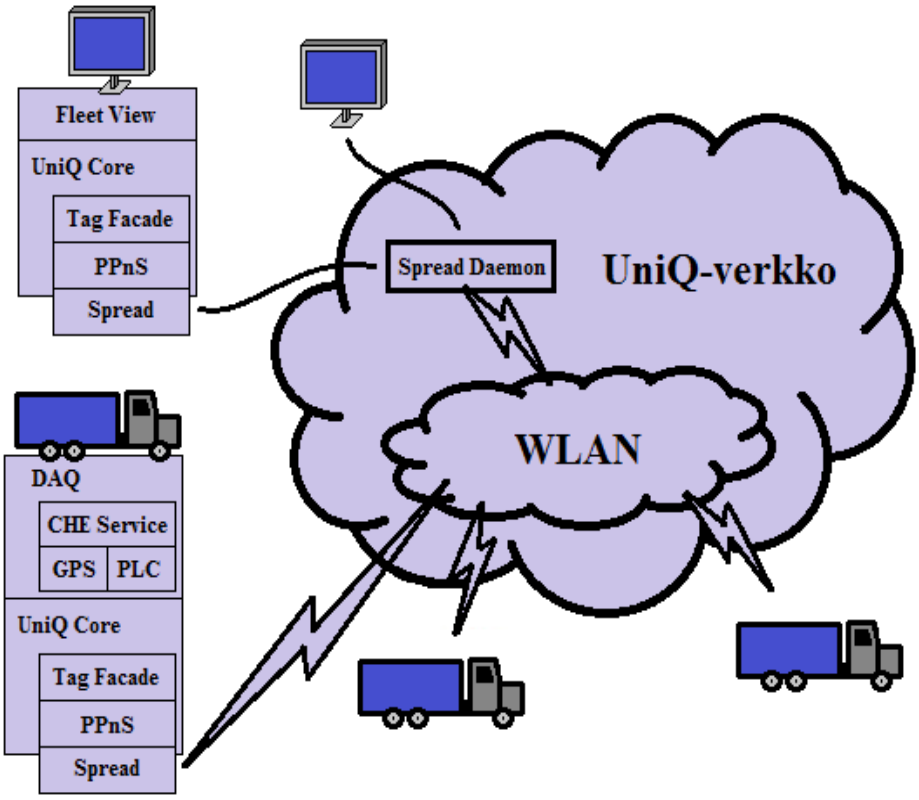
Toinen esimerkki Tag Facaden tarjoamista palveluista on varoitusten lähettäminen. Varoitus voidaan lähettää esimerkiksi lähestyvästä vuosihuollosta, vähäisestä polttoaineen määrästä tai moottoriviasta. Monesti tämätyyppisten viestien on syytä mennä perille ennemmin tai myöhemmin, vaikka virrat katkeaisivat juuri kriittisellä hetkellä. Tästä syystä varoitukset välitetään tavallisesti levyvarmennettuina viesteinä.

Kolmas tyypillinen esimerkki on tiedon kirjoitus, jolla operaattori voi muuttaa etänä esimerkiksi laitteen asetuksia. Tällaiset operaatiot voivat olla myös automatisoituja ja niistä halutaan monesti ilmoitus, kun kirjoittaminen on suoritettu loppuun. Tag Facaden rajapinnan kirjoitusfunktiota voidaan pyytää palautumaan vasta kirjoituksen onnistuttua tai suurimman sallitun kirjoitusajan ylityttyä. Jos tällaisia synkronoituja kirjoitustehtäviä suoritetaan paljon peräkkäin, kaikki ylimääräinen aika, jonka viesti viettää alustan eri kerroksissa, on haitaksi.

2.1.4 Alustan käyttö

Satama-automaatiossa on monia osajärjestelmiä, joilla on omat vastuualueensa. Näiden järjestelmien täytyy kommunikoida keskenään luotettavasti, jotta sataman automaatiojärjestelmä toimii moitteetta. Kalmarin järjestelmässä kommunikointi on toteutettu monilta osin UniQ-alustan avulla. Alustan ydin aluetta ovat kontinkäsittelylaitteilta kerättävän tiedon siirtäminen sitä tarvitseville järjestelmän osille. Yksi tällainen järjestelmän osa on sataman monitorointi- ja hallintatyökalu (Fleet View). Sen avulla operaattorit näkevät sataman tapahtumat sekä kentällä olevien

kontinkäsittelykoneiden tiedot ja tilan. Kuva 4 havainnollistaa järjestelmien välistä kommunikaatiota.



Kuva 4: UniQ-alustan käyttö järjestelmienvälisessä kommunikaatiossa

Kuvan 4 yläosassa olevat näyttöpäätteet kuvaavat monitorointi- ja hallintatyökalujen instansseja (Fleet View, FW). FW käyttää UniQ-alustaa (kuvassa UniQ Core) liittyäkseen UniQ-verkkoon. Kaikki samaan palvelimeen (spread daemon) yhdistäneet laitteet ovat samassa UniQ-verkossa. FW, kuten suurin osa järjestelmän osista on yhteydessä palvelimeen nopealla kiinteällä verkkoyhteydellä (musta käyrä viiva).

Kuvan alaosassa olevissa kontinkäsittelykoneissa on tiedonkeruuohlemisto (kuvassa DAQ). Ohjelmisto kerää tietoa kontinkäsittelykoneen laitteista kuten PLC:ltä (Programmable logic controller) tai GPS-vastaanottimelta (Global Positioning System). Tiedonkeruuohjelmisto käyttää myös UniQ Corea liittyäkseen UniQ-verkkoon. Kontinkäsittelykoneet ovat usein langattoman yhteyden varassa (salama).

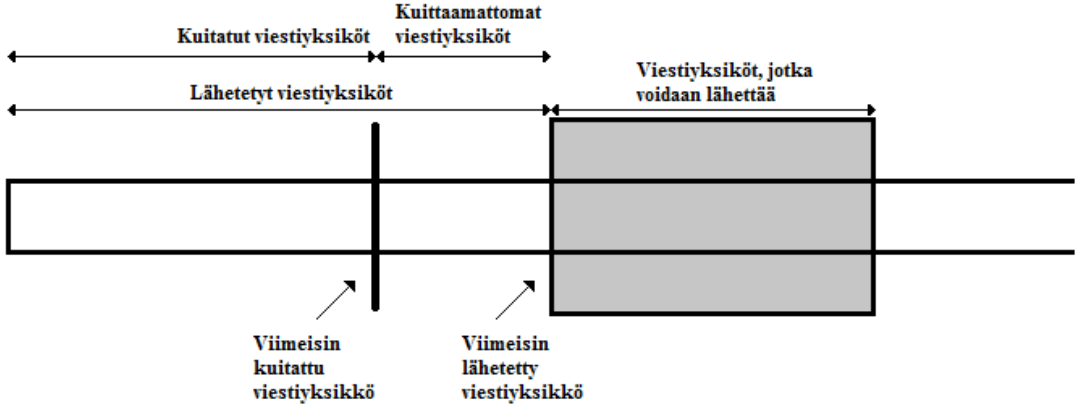
2.2 TCP:n vuonohjaus ja ruuhkanhallinta

Tietoverkkojen ja Internetin tiedonsiirtokyky on rajallinen. Lisäksi verkkojen resurssit täytyy jakaa usean käyttäjän kesken. Tämä aiheuttaa ongelman, kun käyttäjien yhteenlaskettu tiedonsiirtotarve ylittää verkon kapasiteetin. Tällöin verkon reitittimet ylikuormittuvat, eli ne eivät ehdi käsittelemään kaikkia lähetettyjä viestejä. Tällaisissa

tilanteissa reitittimet jättävät osan viesteistä käsittelemättä. Tästä aiheutuu pakettihäviötä, minkä seurauksena verkon tiedonsiirtokyky romahtaa.

Tiedonsiirtoprotokollan toteutus on keskeisessä roolissa, jotta verkon suorituskyky pysyy hyvänä suurenkin kuorman alla. Suurin merkitys on sillä, kuinka lähettäjä kontrolloi omien viestiyksiköiden lähettämistä ja uudelleenlähettämistä. [3, s. 309]

Kuten suurin osa vuonohjausta (eng. flow control) tukevista protokollista, TCP käyttää vuonohjaukseen liukuvan ikkunan mekanismia [3, s. 309]. Ideana on lähettää vain rajoitettu määrä viestejä ilman, että niiden perille menosta on saatu kuittausta. Kuva 5 selventää liukuvan ikkunan ideaa.



Kuva 5: TCP:n lähetysikkuna.[3, s. 311]

Kuvan 5 vasemmassa laidassa ovat yksiköt, jotka on lähetetty ja niihin on saatu kuittaus. Paksun pystyviivan ja harmaan laatikon välissä olevat yksiköt on lähetetty, mutta niiden perille menosta ei ole saatu kuittausta. Harmaan ikkunan sisällä ovat ne viestiyksiköt, joita ei ole vielä lähetetty, mutta ne voidaan lähettää ilman, että yhteenkään viestiyksikköön saadaan kuittausta. Ikkunan oikealla puolella on lisäksi ne viestiyksiköt, jotka voidaan lähettää vasta, kun aiempiin viesteihin on saatu kuittauksia. Huomattavaa on, että ikkunan vasen reuna siirtyy oikealle päin, kun viestejä lähetetään. Oikea reuna sitä vastoin siirtyy oikealle päin, kun kuittauksia vastaanotetaan. Tässäkin on hyvä huomata, että TCP:ssä vastaanottaja voi kuitata viestiyksikön antamatta lähettäjälle lupaa lähettää uusia viestejä. Tätä tapahtuu silloin, kun vastaanottajan viestiyksikköpuskuri on täynnä.

Tiedonsiirtonopeus lähettäjän ja vastaanottajan välillä riippuu lähetysikkunan koosta, tiedonsiirtoviiveestä ja käytettävissä olevasta verkon tiedonsiirtokaista [3, s. 313]. Tietoa ei luonnollisesti voida siirtää nopeammin, kuin käytössä oleva tiedonsiirtokaista sallii. Lähetysikkunan koko voi myös olla rajoittava tekijä, jos se on liian pieni suhteessa tiedonsiirtokaistan leveyteen ja verkossa on riittävän suuri viive. Tämä aiheutuu silloin, jos kaikki lähetysikkunassa olevat viestiyksiköt on lähetetty ja ensimmäisten viestiyksiköiden kuittaus ei ole ehtinyt saapua.

Ruuhkanhallinta on monimutkainen asia ja siitä on tehty suuri määrä tutkimuksia vuosikymmenten aikana. Asian tekee monimutkaiseksi kolme tekijää. Ensinnäkin TCP toimii IP:n päällä. IP on yhteydetön protokolla, mikä ei takaa viestiyksiköiden perille menoa sen paremmin, kuin ruuhkanhallintaakaan. Toisekseen TCP pystyy tarkkailemaan ainoastaan viestiyksiköiden lähetystä ja kuittauksien vastaanottamista. Kaikki muu pitää pystyä laskemaan näistä tiedoista. Komanneksi TCP:n täytyy otta huomioon myös muut verkon käyttäjät. Se ei voi pyrkiä itsekkäästi käyttämään mahdollisimman suurta osaa verkon kapasiteetista.[3, s. 322]

2.2.1 Viestiyksikön uudellenlähetysajan laskeminen

Kuittausviive (eng. round-trip time, RTT) on aika, joka kestää viestyksikön lähettämisestä sen kuittaamisen vastaanottamiseen. TCP:ssa viestiyksiköiden uudelleen lähettämistä kontrolloi lähettäjä. Jos viestiyksikköön ei saada kuittausta riittävän ajan kuluessa, sen oletetaan hävinneen ja lähetetään uudelleen. Kuittausviive ei ole vakio, vaan se voi vaihdella paljonkin. Yksi vaikuttava tekijä on TCP:n mahdollisuus olla kuittaamatta jokaista viestiyksikköä erikseen — se voi kuitata useamman viestiyksikön kerrallaan [3, s. 321]. Kuittausviiveellä on vaikutusta sekä viestiyksiköiden uudelleenlähetykseen että ikkunan kokoon.

TCP:ssä viestiyksikkö lähetetään uudelleen, jos siihen ei ole saatu kuittausta riittävän ajan kuluessa. Vakioaika ei sovellu tähän tarkoitukseen kolmesta syystä. Ensinnäkin TCP:n ei tarvitse kuitata jokaista viestiyksikköä erikseen. Toisekseen kuittauksen vastaanottaja ei voi tietää, onko kuittaus alkuperäiseen viestiyksikköön vai uudelleenlähetettyyn viestiyksikköön. Kolmannekseen Internetin olosuhteet voivat muuttua yhtäkkiä. [3, s. 316]

Jos uudelleenlähetysaika on liian pieni, viestejä lähetetään turhan usein ja ne lisäävät verkon ruuhkaa. Liian suuri aika sen sijaan aiheuttaa turhia viiveitä, jos viestiyksiköitä katoaa.

Yksi käytetty menetelmä laskea sopiva uudelleenlähetysaika on Jacobsonin algoritmi. Siinä ideana on laskea kuittausviiveen liukuvaa painotettua keskiarvoa sekä liukuvaa painotettua keskihajontaa.

$$
\begin{aligned}
SRTT(K+1) &= (1-g)\times SRTT(K) + g \times RTT(K+1) \\
SERR(K+1) &= RTT(K+1) - SRTT(K) \\
SDEV(K+1) &= (1-h)\times SDEV(K) + h \times |SERR(K+1)|
\end{aligned}
\qquad \textbf{(1)}\ [3, \text{s. } 327]
$$

$$
RTO(K+1) = SRTT(K+1) + f \times SDEV(K+1)
$$

Kuittausviiveen ennuste (eng. smoothed round-trip time estimate, *SRTT*) saadaan laskemalla painotettu keskiarvo viimeisimmistä kuittausviiveistä. Vakio g määrittää, missä suhteessa arvoja painotetaan. Uudelleenlähetyksen aika (eng. retransmission timeout, *RTO*) saadaan laskemalla yhteen kuittausviiveen ennuste (eng. smoothed

round-trip estimate, *SRTT*) ja vakiolla (*f*) kuittauviiveen keskihajonnan ennuste (eng. moothed roundtrip standard deviation estimate , *SDEV*).

Jacobson esitti kaavan 1 vakioille arvot:

$g = 1/8 = 0.125$

$h = 1/4 = 0.25$

$f = 2$ (myöhemmin $f = 4$) [3, s. 328].

Viralliseksi suositukseksi *RTO*:lle on muotoutunut

$$RTO(K+1) = SRTT(K+1) + MAX[G, f \times SDEV(K+1)], \quad \textbf{(2)}\ [3, \text{s. } 328]$$

missä *G* on sopiva pieni luku. Toisin sanoen keskihajonnan ollessa nolla tai hyvin pieni, se on vähintään vakion *G* verran.[3, s. 328]

2.2.2 Lähetysikkunan koon hallinta

Tehokkaan uudelleenlähetysajan laskemisen lisäksi TCP:n lähetysikkunan koon hallinnalla on suuri merkitys verkon ruuhkautumisen estämisessä. Tällaisia ovat muun muassa maltillinen aloitus (eng. slow start), ikkunan pienentäminen ruuhkatilanteessa (eng. dynamic window sizing on congestion), nopea uudelleenlähetys (eng. fast retransmit) ja nopea palautuminen (eng. fast recovery).[3, s. 331]

Yksi mahdollinen vaihtoehto määrittää lähetysikkunan aloituskoko, on yrittää arvata se. Tämä on riskialtista, sillä liian suuri alkuikkuna voi ruuhkauttaa koko verkon ennen, kuin huomaa verkon ylikuormiittuneen. Jacobson suosittelee aloittamaan lähettämisen maltillisesti.[3, s. 331]

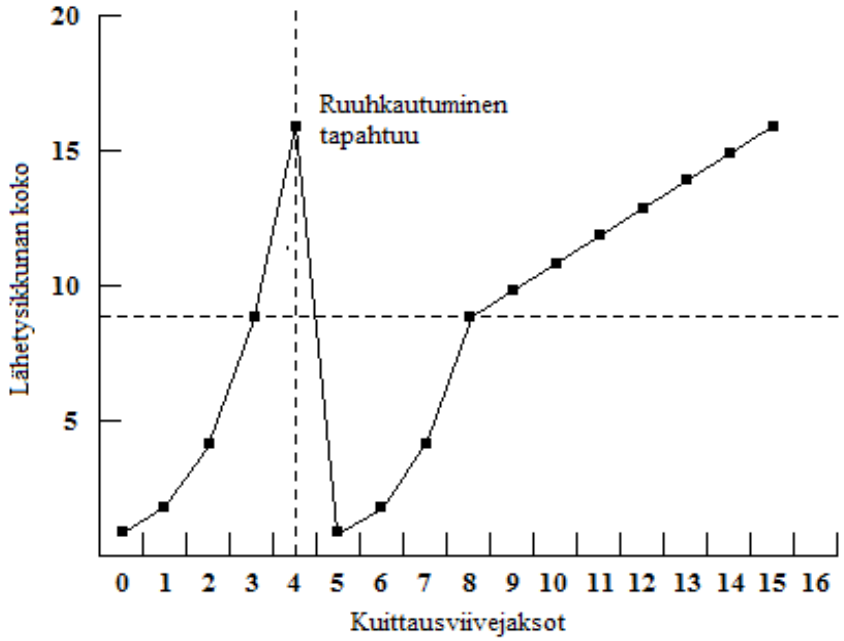
Kun uusi yhteys avataan, lähetysikkunan kooksi asetetaan 1. Toisin sanoen TCP voi lähettää yhden viestiyksikön, jonka jälkeen sen täytyy odottaa kuittausta ennen seuraavien viestiyksiköiden lähettämistä. Jokaista saapuvaa kuittausviestiä kohden ikkunan kokoa kasvatetaan yhdellä.[3, s. 332]

Maltillisen aloituksen tarkoituksena on tunnustella verkon kapasiteettia ja vältetään tilanne, jossa valmiiksi ruuhkautuneeseen verkkoon lähetetään liikaa viestejä. Ikkunan koko kasvaa, kunnes kuittausviestejä alkaa hävitä.[3, s. 332]

Huomattavaa on, että ikkunan koko kaksinkertaistuu kuittausviiveen välein. Ikkunan kokoa kasvatetaan yhdellä jokaista kuittausviestiä kohden. Aluksi lähetetään yksi viesti, johon saadaan yksi kuittaus. Seuraavaksi kaksi viestiä, joihon saadaan yhteensä kaksi kuittausta. Sitten viestejä voidaan lähettää jo neljä.[3, s. 332]

Rauhallisella aloituksella saadaan nopeasti selville järkevä lähetysikkunan koko. Kuittausviestin häviäminen on merkki verkon ruuhkautumisesta. Yksi vaihtoehto olisi asettaa lähetysikkunan koko yhdeksi ja toteuttaa rauhallinen aloitus uudelleen, kunnes havaitaan ruuhkautumista. Tämä ei toimi hyvin käytännössä sillä ikkunan eksponentiaalinen kasvu voi pahentaa verkon ruuhkautumista.[3, s. 332]

Parempi vaihtoehto on käyttää seuraavanlaista menetelmää. Kun havaitaa ruuhkautumista, lähetysikkunan koko asetetaan yhdeksi. Tämän jälkeen noudatetaan rauhallista aloitusta, kunnes ikkunan koko on puolet siitä, kun ruuhkautuminen havaittiin edellisellä kerralla. Tämän jälkeen ikkunan kokoa kasvatetaan yhdellä jokaista kuittausviivettä kohden.[3, s. 332] Kuva 6 selventää tilannetta.



Kuva 6: Maltillinen aloitus ja ruuhkan välttäminen. [3, s. 335]

Huomattavaa on, että saman lähetysnopeuden saavuttamiseen menee ensimmäisellä kerralla neljä kuittausviivejaksoa ja toisella kertaa 11 [3, s. 334].

2.2.3 TCP Cubic

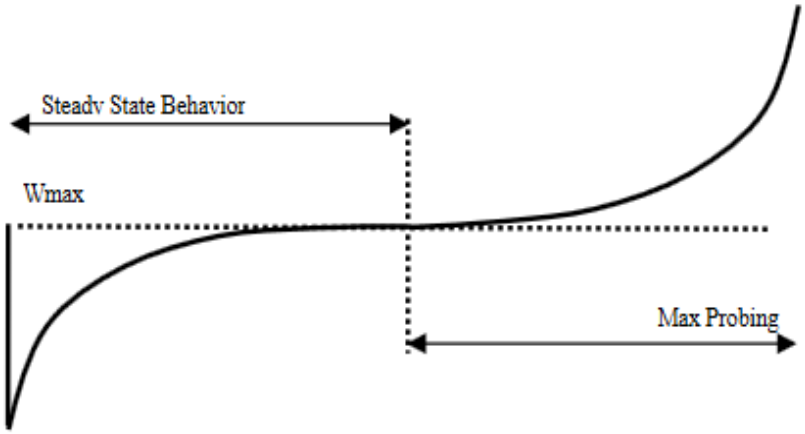
Myöhemmät tutkimukset ja toteutukset TCP:n lähetysikkunan koon hallinnasta ovat tuoneet parempia tapoja saavuttaa optimaalinen lähetysnopeus. Yksi tällainen muunnos on TCP Cubic.

TCP Cubic:n ikkunankasvatus-funktio (eng. window growth function) on nimensä veroinen eli se liittyy kuutioon (eng. cubic) [4, s. 2]. Ikkunan kokoa ei kasvateta vakion verran vaan sille on monimutkaisempi funktio:

$$W_{cubic} = C(t-K)^3 - W_{max}\,, \qquad \textbf{(3)}\ [4,\ \text{s. } 2]$$

missä C on skaalauskerroin, t aika viimeisestä ikkunan pienentämisestä, W_{max} ikkunan koko juuri ennen sen pienentämistä ja $K = \sqrt[3]{W_{max}\beta / C}$, missä β on vaiko, joka määrittelee, kuinka paljon ikkunaa pienennetään ruuhkatilanteissa (ikkunaa pienennetään bW_{max} verran).[4, s. 2]

Cubic:ssa ikkunan kasvatus ei ole sidottu kuittausviiveeseen, vaan sitä kasvatetaan kuluneen ajan mukaan. Tästä johtuen Cubic ei kuormita verkkoa liikaa riippumatta siitä, onko verkon viive suuri tai pieni.[4, s. 1] Kuvassa 7 on esitetty graafisesti Cubic:n ikkunankasvatus-funktio heti ruuhkanhavaitsemisen jälkeen.



Kuva 7: TCP Cubic:n ikkunankasvatus-funktio, [6, s. 2]

Ikkunan pienentämisen jälkeen sen kokoa kasvatetaan nopeasti kohti ruuhkautumistasoa. Kasvatusnopeutta hidastetaan tason lähestyessä. W_{max}:n kohdalla ikkunan kasvatusnopeus on lähestulkoon nolla, jonka jälkeen se alkaa taas kasvaa kiihtyvällä vauhdilla.[4, s. 2]

Toinen muunnos, TCP Westwood on suunniteltu langattomia verkkoja varten. Langattomissa verkoissa viestiyksiköitä häviää satunnaisesti, vaikka ruuhkautumista ei tapahtuisi.[5, s. 465]

Westwoodin keskeisenä ideana on arvioida paketin häviämishetkellä olevaa tiedonsiirtokaistaa kuittausviestien perusteella. Jos yksittäisiä kuittausviestejä jää saapumatta, Westwood pienentää lähetysnopeutta laskemansa tiedonsiirtokaista-arvion pohjalta.[5, s. 447]

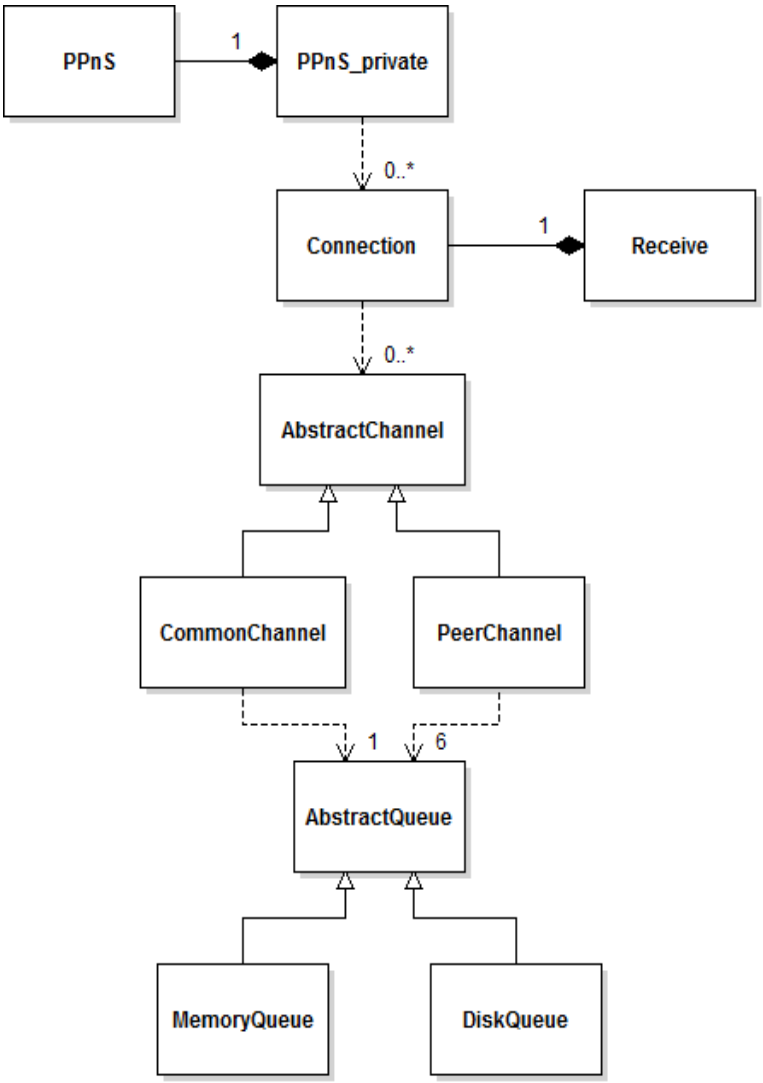
Tämä on röyhkeämpi tapa toimia, kuin perinteisillä algoritmeilla. Käytännön mittauksissa on havaittu Westwoodin ryöstävän kaistaa muilta, mutta se ei kuitenkaan aiheuta muiden nälkiintymistä.[5, s. 447]

3 PPNS

Tässä luvussa käydään läpi PPnS:n toiminta ja rakenne ennen työn aloittamista. Apuna käytetään luokka- ja ajoituskaavioita. Erityistä huomiota kiinnitetään siihen, miten viestit lähetetään ja vastaanotetaan.

3.1 Toiminta

PPnS:n toiminta on helpoin ymmärtää luokkakaavion avulla. Kuvassa 8 on PPnS:n luokkakaavio työn aloitushetkellä.



Kuva 8: PPnS-tason luokkakaavio

PPnS on luokkakirjasto, joten sille on toteutettu erikseen julkinen (PPnS) ja yksityinen (PPnS_private) -rajapinta. Julkisen rajapinnan toteuttava luokka luo yksityisen rajapinnan toteuttavan luokan ajonaikana ja käsittelee sitä osoittimen läpi. Tästä on se hyöty, että kirjaston yksityistä osaa voidaan päivittää vapaasti jälkikäteen ja sitä käyttäviä ohjelmia ei tarvitse kääntää uudelleen.

Connection-luokka kuvaa yksittäistä yhteyttä daemoniin. Yleensä yksi yhteys riittää, mutta joissakin tapauksissa yhteyksiä tarvitaan useampi. Useamman yhteyden käytöstä on hyötyä esimerkiksi silloin, kun verkon pakettihäviö ja viive ovat suuria. Koska jokainen Connection-luokan instanssi luo yhden TCP-yhteyden daemoniin, tiedonsiirtonopeus voi olla parhaimmillaan suoraanverrannollinen yhteyksien määrään. Connection-luokasta on luotava useampia instansseja myös silloin, jos halutaan olla yhteydessä useampaan daemoniin.

AbstractChannel-luokka kuvaa laitteiden välisiä yhteyksiä. Kyseessä on abstrakti kantaluokka, joten siitä ei voida luoda instanssia. PeerChannel ja CommonChannel perivät AbstractChannel-luokan ja toteuttavat puuttuvat metodit.

PeerChannel kuvaa kahden laitteen välistä yhteyttä. Tämä on yleisimmin käytetty kanava. CommonChannel:n tehtävänä on huolehtia sykeviestikanavan liikenteestä, joten jokainen Connection-luokan instanssi sisältää yhden CommonChannel-luokan instanssin. CommonChannel-luokka sisältää myös ryhmäviestintään tarvittavan toiminnallisuuden. Ryhmäviestintä on myöhemmin siirretty omaan luokkaansa muiden kehittäjien toimesta.

AbstractQueue-luokka kuvaa viestijonoa, jota käytetään lähtevien viestien puskurointiin viestin välitystakuun mukaan. Kyseessä on jälleen abstrakti kantaluokka ja sen toteuttavat MemoryQueue ja DiskQueue.

MemoryQueue toteuttaa työmuistissa säilytettävän jonon. Tämä jono säilyy, kunnes kanava suljetaan käyttäjän toimesta tai koko PPnS instanssi tuhotaan. Tämä tapahtuu esimerkiksi silloin, kun laitteesta katkaistaan virrat.

DiskQueue toteuttaa massamuistissa säilytettävän jonon. Jonon sisältö säilyy vaikka kanava suljettaisiin tai virrat katkeaisivat.

Viestien vastaanottamisesta huolehti Receive-luokka. Luokkaa ajetaan omassa säikeessän, jotta viestien lähettäminen ja vastaanottaminen häiritsisivät mahdollisimman vähän toisiaan. On hyvä huomata, että viestijonoja ei käytetä enään viestejä vastaanotettaessa vaan ne käsitellään saman tien.

Seuraavaksi käydään läpi viestien lähetyksen ja vastaanottamisen logiikka ajoituskaavioiden avulla siinä muodossa, kuin ne ovat olleet aloitushetkellä. Nämä asiat on käsitelty kappaleissa 3.2 ja 3.3.

3.2 Viestien lähettäminen

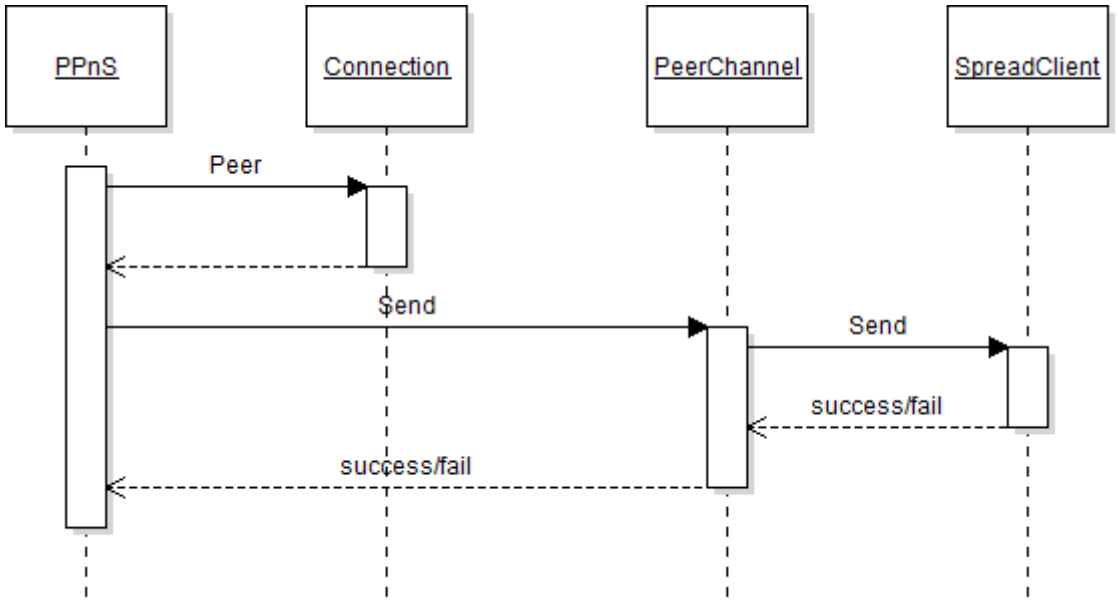
PPnS sisältää useita takuita viestien välittämiseen. Nämä takuut on käsitelty kappaleessa 2.1.2 ja niistä on tehty kooste taulukkoon 1.

Viestit, joille ei anneta mitään takuuta, voidaan lähettää välittömästi ja unohtaa saman tien. Varmentamattomat, muistivarmennetut ja levyvarmennetut viestit tallennetaan erillisiin jonoihin ja vuorontaja huolehtii niiden lähettämisestä.

Esimerkeissä käsitellään vain PeerChannel-luokkaa, koska sillä ei ole oleellista eroa CommonChanneliin suorituskyvyn näkökulmasta. Tästä syystä CommonChannelin tarkempi käsittely ohitetaan tässä työssä.

3.2.1 Välittömät viestit

Välittömien viestien lähettäminen on hyvin yksinkertaista ja suoraviivaista. Kuvassa 9 on välittömän viestin lähettämisen ajoituskaavio.



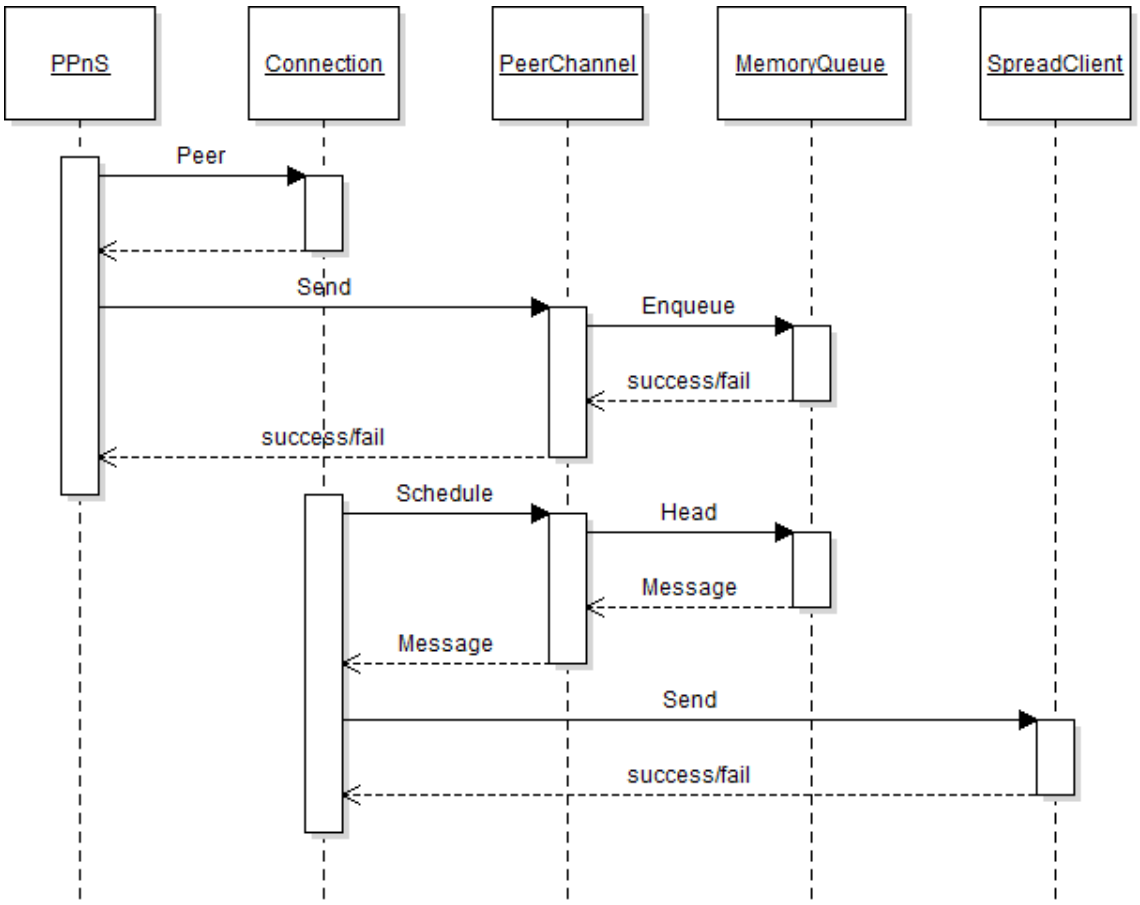
Kuva 9: Välittömien viestien lähetyksen ajoituskaavio.

Lähetys käynnistyy, kun käyttäjä, eli Tag Facade kutsuu PPnS:n julkisen rajapinnan Send-metodia. Connection-luokka pitää kirjaa laitteista, joihin on luotu yhteys. Tästä syystä PPnS-luokka pyytää tarvittavan PeerChannel-instanssin osoittimen kutsumalla Connection-luokan Peer-metodia. Tämän jälkeen PPnS-luokka kutsuu saadun instanssin Send-metodia. PeerChannel tutkii viestin perillemenotakuun. Koska kyseessä on välitön viesti PeerChannel luokan instanssi kutsuu SpreadClientin Send-metodia. Viestin lähetyksen onnistumistieto välitetään PeerChannelin ja PPnS-luokan kautta Tag Facadelle. Lähettäminen voi epäonnistua esimerkiksi silloin, kun yhteys on katkennut jostain syystä.

3.2.2 Varmentamattomat ja muistivarmennetut viestit

Varmentamattomat ja muistivarmennetut viestit tallennetaan molemmat omiin johoihinsa. Jonot sijaitsevat nopeassa keskusmuistissa ja niiden lähettämisestä huolehtii erillinen vuorontaja.

Varmentamattomat viestit voidaan poistaa jonosta heti lähetyksen jälkeen, koska niiden perillemenoa ei varmisteta. Varmentamattomien viestien etu välittömiin viesteihin on siinä, että ne puskuroidaan yhteyskatkon aikana ja lähetetään yhteyden palattua. Kuva 10 esittää varmentamattomien viestien lähettämistä.



Kuva 10: Varmennettujen viestien lähetyksen ajoituskaavio.

Tag Facade kutsuu PPnS:n Send-metodia, jolloin PPnS-luokka pyytää Connection-luokalta oikean PeerChannelin instanssin. Tämän jälkeen PPnS-luokka kutsuu saadun instanssin Send-metodia. Varmentamattomien viestien jono on MemoryQueue tyyppinen. Viesti lisätää varmentamattomien viestien jonoon kutsumalla MemoryQueue-luokan Enqueue-metodia. Tieto siitä, onnistuiko viestin lisääminen jonoon, välitetään PeerChannelin ja PPnS:n kautta Tag Facadelle.

Varsinaisesta viestin lähettämisestä huolehtii erillinen vuorontaja. Vuorontaja on toteutettu Connection-luokassa. Kun vuorontaja näkee hyväksi lähettää viestin, se pyytää viestin PeerChannelilta kutsumalla sen Schedule-metodia. PeerChannel hakee

viestin varmentamattomien viestien jonosta kutsumalla MemoryQueue-luokan Head-metodia. Viesti välitetään PeerChannelin kautta Connection luokan instanssille, joka kutsuu SpreadClient:n Send-metodia.

Ainoa käytännön ero varmentamattomien ja muistivarmennettujen viestien välillä on se, että muistivarmennetut viestit täytyy säilyttää myös lähettämisen jälkeen. Ajoituskaavio on muistivarmennettujen viestien kohdalla samanlainen. Erona on se, että kutsuttaessa MemoryQueuen Enqueue-metodia, suoritetaan tarkastus sille, onko jonon maksimipituus ylittynyt. Jos maksimipituus ylittyy, jonosta poistetaan vanhin viesti. Varmentamattomat viestit voitaisiin poistaa heti lähetyksen jälkeen, mutta niitäkin ryhdytään poistamaan vasta jonon täytyttyä.

PPnS:ssä ei ole erillistä kuittaustoimintoa vaan vastaanottajan täytyy erikseen pyytää lähettäjää lähettämään haluttu paketti uudelleen. Muistivarmennetut viestit on varustettu juoksevalla järjestysnumerolla. Vastaanottaja havaitsee paketin häviämisen, kun jokin järjestysnumero jää väliin. Tällöin vastaanottaja pyytää lähettäjää jatkamaan lähettämistä ensimmäisestä väliinjääneestä järjestysnumerosta. Tällainen toiminnallisuus on riittävä, koska TCP varmistaa viestien perillemenon sekä oikean järjestyksen. Näin vältetään turhien kuittausviestien lähetys.

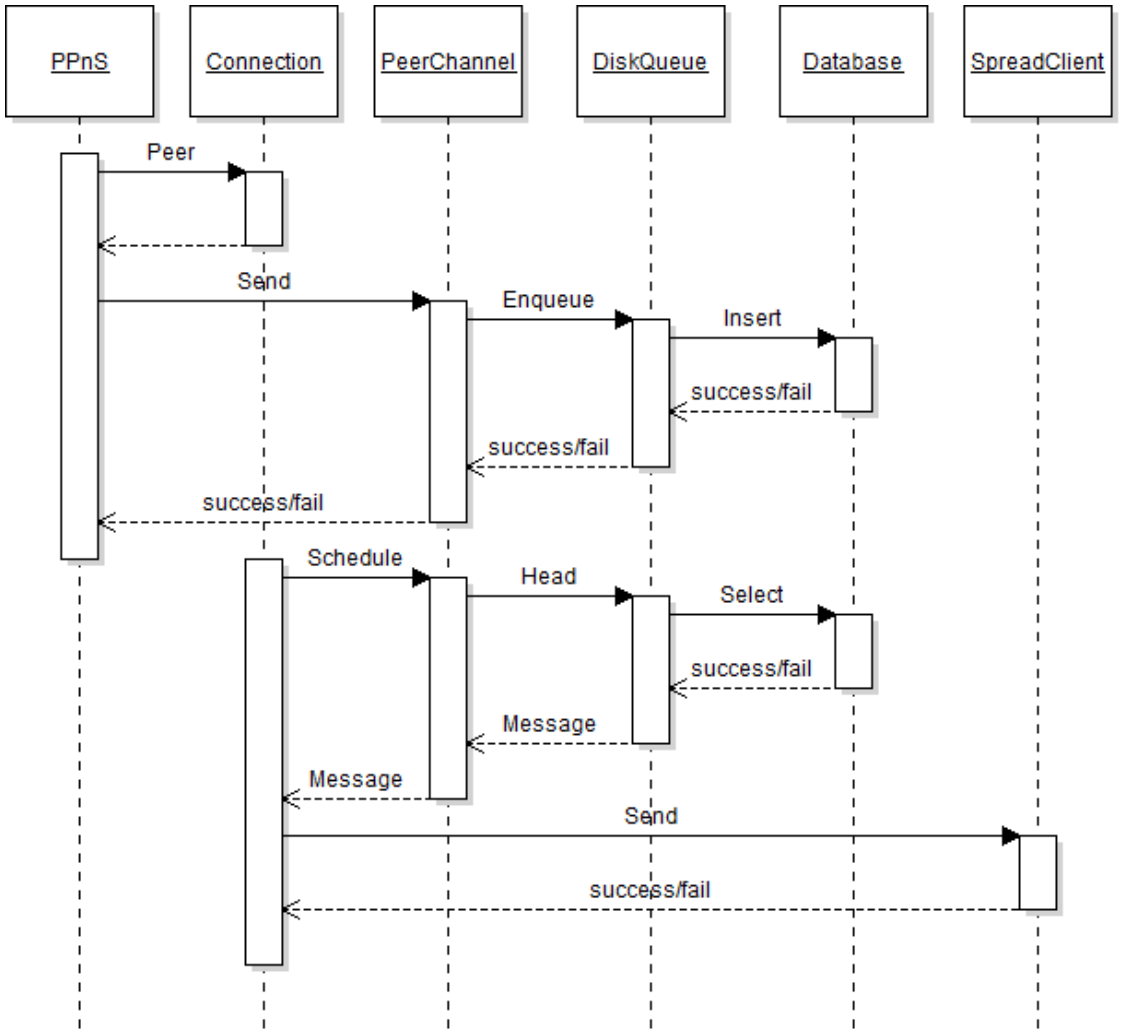
Koska varmentamattomien ja muistivarmennetujen viestien lähettämisessä ainoa ero on se, milloin viesti voidaan poistaa, niiden voidaan olettaa olevan yhdenvertaisia suorituskyvyn kannalta.

3.2.3 Levyvarmennetut viestit

Levyvarmennetut viestit tallenetaan jonoon, kuten varmentamattomat ja muistivarmennetut viestit. Viestien lähettämisestä huolehtii vuorontaja.

Viestien lähettäminen ja uudelleen lähettäminen tapahtuvat samalla tavalla, kuin muistivarmennettujen viestien kohdalla — vastaanottajan täytyy erikseen pyytää uudelleenlähetys. Keskeisenä erona suorituskyvyn kannalta on se, että jono sijaitsee massamuistissa. Tämä tarkoittaa sitä, että viestien käsittely on huomattavasti hitaampaa, kuin muistivarmennettujen viestien käsittely.

Levyjono on toteutettu käyttäen SQLite-tietokantaa. Tietokannan käyttöä puoltaa takuu tiedon säilymisestä sekä helppo toteutus. Levyjonon tarkoituksena on varmistaa, että viestit eivät katoa, vaikka virrat katkeaisivatkin. Jos saman toiminnallisuuden toteuttaisi tiedostoilla, aikaa kuluisi moninkertaisesti ja virheiden todennäköisyys olisi huomattavasit suurempi. Kuvassa 11 on kuvattu levyvarmennetun viestin lähettäminen ajoituskaaviolla.



Kuva 11: Levyvarmennetun viestin lähetyksen ajoituskaavio.

Levyvarmennettujen viestien lähetys on hyvin lähellä muistivarmennettujen viestien lähetystä. Suurimpana erona on tietokannan käyttö.

Tag Facade kutsuu PPnS:n Send-metodia, jolloin PPnS-luokka pyytää Connection-luokalta oikean PeerChannelin instanssin. Tämän jälkeen PPnS-luokka kutsuu saadun instanssin Send-metodia. Viestin lisätää varmentamattomien viestien jonoon, kutsumalla DiskQueuen Enqueue-metodia. DiskQueue-luokka lisää viestin tietokantaan (Database), joka sijaitsee massamuistissa. Tieto siitä, onnistuiko viestin lisääminen jonoon välitetään PeerChannelin ja PPnS:n kautta Tag Facadelle.

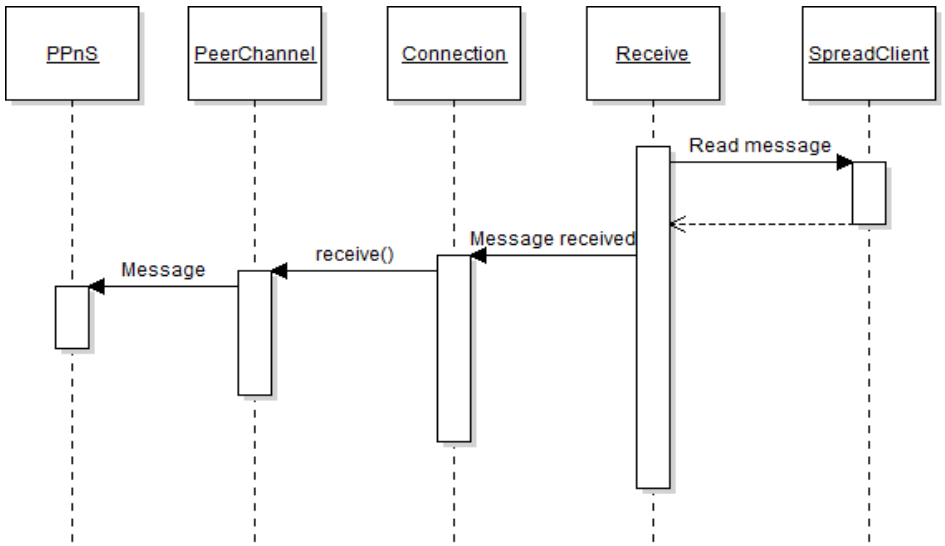
Viestin lähettämisestä huolehtii erillinen vuorontaja samalla tavalla, kuin muistivarmennettujen viestien lähettämisestä. Kun vuorontaja katsoo hyväksi lähettää viestin, se pyytää viestin PeerChannelilta kutsumalla sen Schedule-metodia. PeerChannel hakee viestin levyvarmennettujen viestien jonosta kutsumalla DiskQueue-luokan Head-metodia. DiskQueue-luokka hakee viestin vastaavasti tietokannasta. Viesti

välitetään PeerChannelin kautta Connection luokan instanssille, joka kutsuu SpreadClient:n Send-metodia.

Tietokannassa säilötään lähetettyjä viestejä samasta syystä, kuin muistivarmennettuja viestejä. Levyvarmennettujen viestien jonolla on samalla tavalla suurin sallittu pituus, jonka jälkeen vanhin viesti poistetaan. Jonon pituuden tarkastus ja viestin poisto tehdään DiskQueue-luokan Enqueue-metodissa samalla tavalla, kuin muistivarmennettujen viestien kohdalla.

3.3 Viestien vastaanottaminen

Viestien vastaanottamisessa ei ole oleellisia eroja eri takuiden omaavien vistien välillä. Suurin ero on siinä, että muisti- ja levyvarmennetuista viesteistä tarkastetaan järjestysnumero. Jos järjestysnumero ei ole oikea, viesti hylätään ja lähetetään uudelleenlähetyspyyntö. Kuva 12 esittää viestien vastaanottamisen ajoituskaaviota.



Kuva 12: Viestien vastaanottamisen ajoituskaavio.

Viestien vastaanottaminen tapahtuu omassa säikeessä. Toisin sanoen kaikki Receive-luokan toiminnallisuus ajetaan erillisessä säikeessä.

Receive-luokan instanssi tarkastaa viiden millisekunnin välein SpreadClientiltä, onko uusia viestejä saapunut. Kun uusi viesti saapuu, Receive-luokan instanssi lukee viestin SpreadClientiltä ja välittää sen Connection- ja PeerChannel luokkien kautta PPnS-luokalle, joka puolestaan välittää viestin eteenpäin Tag Facadelle.

4 SUORITUSKYVYN MITTAUS

Tässä luvussa käydään läpi mittauksen tavoitteet, käytetyt menetelmät, mittausympäristö sekä mittauksissa käytetty ohjelmisto. Luvussa esitellään myös alustan suorituskyky lähtötilanteessa.

4.1 Mittauksen tavoitteet

Tässä työssä mittausten tavoitteena on selvittää tiedonsiirtoalustan viestien välityskyky, viestien lähettämisestä ja vastaanottamisesta aiheutuva prosessorikuorma sekä viestin välittämiseen kuluva aika. Näitä suureita voidaan mitata monella tavalla, ja tässä työssä käytetyt menetelmät on kuvattu kappaleessa 4.2

Työssä käsitellään kahta erillistä ongelmaa ja niiden ratkaisussa vaadittavat mittaustarkkuudet eroavat toisistaan. Suorituskyvyn optimoinnissa oleellista on, että pystytään osoittamaan tehtyjen muutosten vaikutuksen suorituskykyyn, joten systemaatisella virheellä ei ole merkitystä. Maksimaalisen suorituskyvyn selvittämisessä mittausympäristön aiheuttama systemaattinen virhe tulee minimoida, jotta saadut tulokset kuvaavat todellisuutta. Tässä mielessä suorituskyvyn mittaukseen käytettävien menetelmien vaatimukset kattavat myös optimointiongelman vaatimukset.

Mittauksissa keskitytään PPnS-tason suorituskykyyn, mutta Spread Toolkitille tehdään vastaavat mittaukset, jotta saadaan selville PPnS-tason aiheuttama kuorma. Tag Facaden osuus kommunikaatiossa on minimaalinen ja sieltä ei löydy vastaavia optimointikohteita, kuin PpnS:stä. Lisäksi sen kunnollinen käsittely ei olisi mahtunut tämän työn mittakaavaan, joten se rajattiin pois.

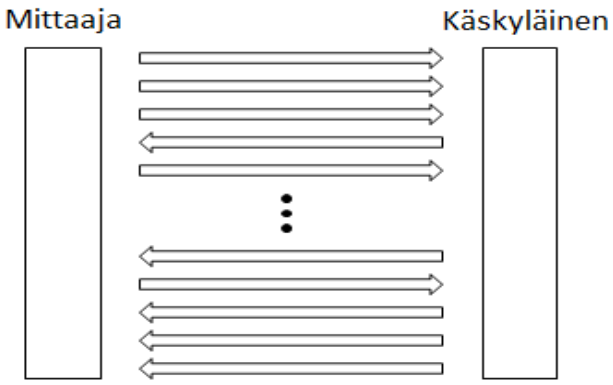
4.2 Menetelmät

Kaikki käytetyt menetelmät on valittu siten, että ne olisi helppo toteuttaa, mutta antaisivat silti riittävän tarkan ja luotettavan tuloksen. Jokaisessa menetelmässä käytetään kahta ohjelmaa. Ohjelmista käytetään nimiä mittaaja ja käskyläinen.

Käskyläisen tehtävänä on palvella mittaajaa eli toteuttaa sen pyynnöt. Mittaajan tehtävänä on koordinoida testaaminen ja mitata halutut arvot. Kunkin suorituskykyyn vaikuttavan tekijän mittauksessa käytetyt menetelmät on kuvattu omissa alaluvuissaan.

4.2.1 Tiedonsiirtonopeus

Tiedonsiirtonopeuden mittaamisessa käytetään menetelmää, jossa mittaaja lähettää viestejä käskyläiselle, joka puolestaan lähettää välittömästi samat viestit takaisin mittaajalle. Kuva 13 selventää asetelmaa.



Kuva 13: Mittaaja lähettää viestejä käskyläiselle, joka lähettää samat viestit takaisin mittaajalle.

Mittaaja lähettää viestejä käskyläiselle, joka vastaa jokaiseen viestiin lähettämällä vastaanottamansa viestin takaisin mittaajalle. Mittaaja odottaa, että käskyläinen vastaa kaikkiin lähetettyihin viesteihin ja mittaa tähän prosessiin kuluneen ajan. Kun paketit lähetetään mahdollisimman nopeasti ja niihin vastataan välittömästi, saadaan mitattua suurin mahdollinen tiedonsiirtonopeus. Lähetettävien viestien koko vaihtelee ja pakettikoot on kerrottu tulosten yhteydessä. Menetelmän etuna on, että se vastaa hyvin käytännön tilannetta, jossa viestejä liikkuu molempiin suuntiin. Testit ajettiin käyttäen Linuxin oletus ruunkanohjausikkunaa (TCP Cubic) [6].

4.2.2 Prosessorikuorma

Prosessorin käytön osalta yksi kiinnostava tieto on, kuinka paljon prosessoriaikaa tarvitaan viestien lähettämiseen ja vastaanottamiseen. Toinen kiinnostava tieto on, kuinka suuri prosessorin käyttö on silloin, kun viestejä ei lähetetä tai vastaanoteta.

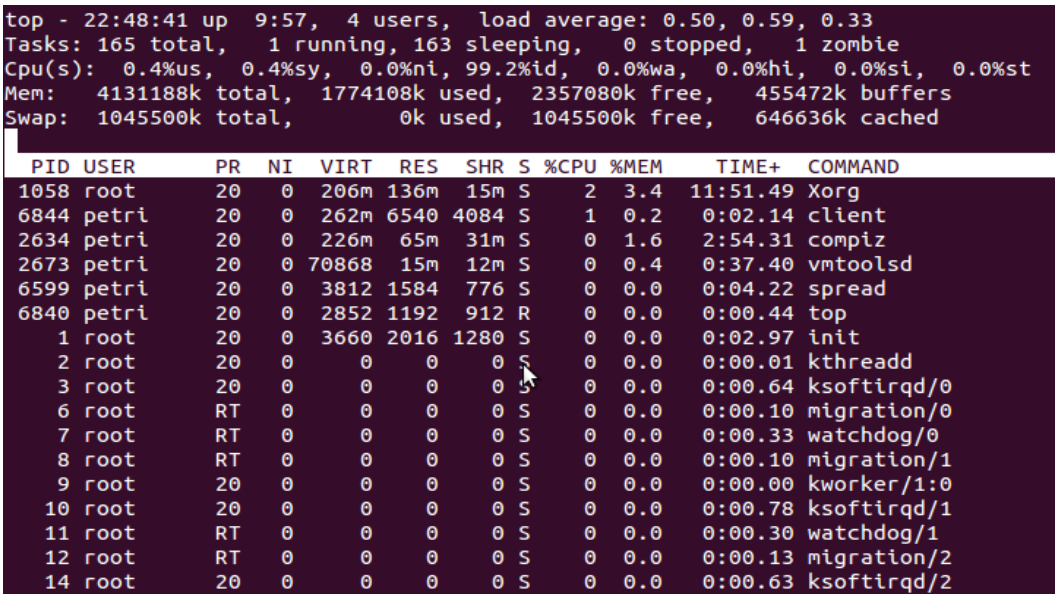
Prosessorikuorman mittaamisessa käytetään Linux-käyttöjärjestelmän tarjoamaa tietoa ohjelman prosessorin käytöstä. Tieto ei ole absoluuttisen tarkka, joten menetelmää käytetään vain osoittamaan, miten PPnS:ään tehdyt muutokset vaikuttavat prosessorikuormaan. Viestikohtainen prosessorikuorma saadaan laskettua ajamalla vastaavanlainen testi, kuin tiedonsiirtonopeuden testaamisessa, mutta ajan sijaan mitataan ohjelman käyttämää prosessoriaikaa. Kaikki prosessorikuormat mitataan A-koneelta ja niissä on mukana viestien lähettämiseen sekä vastaanottamiseen kulunut prosessoriaika.

Jotta testiohjelman prosessorin käyttö saadaan eliminoitua, vähennetään testituloksesta ensin tyhjällä kuormalla ajettu testiohjelman prosessorinkäyttö ja jaetaan saatu erotus lähetettyjen viestien määrällä kaavan 4 mukaisesti.

$$P_m = \frac{P_l - P_i}{n} \quad , \tag{4}$$

missä P_m on yhden viestin lähettämiseen ja vastaanottamiseen kuluva prosessoriaika, P_l kuorman kanssa ajetun testin prosessorikäyttö, P_i ilman kuormaa ajetun testin prosessorikäyttö ja n lähetettyjen viestien määrä. Kaavaa 4 voidaan soveltaa myös silloin, kun tuloksesta halutaan eliminoida alempien kerrosten prosessorinkäyttö.

Toinen kiinnostava asia on PPnS:n prosessorin käyttöaste silloin, kun varsinaista liikennettä ei ole. Tämän mittaamiseen käytetään Linuxin top-komentoa. Kuvassa 14 on kuvakaappaus top-komennon antamasta tuloksesta.



Kuva 14: Top-ohjelman näkymä.

Kunkin ohjelman prosessorin käyttöaste voidaan lukea CPU% sarakkeesta. Arvo on prosentteina yhdestä ytimestä. Jos tietokoneen prosessorissa on esimerkiksi neljä ydintä, prosessorin käyttöaste voi olla parhaimmillaan 400%.

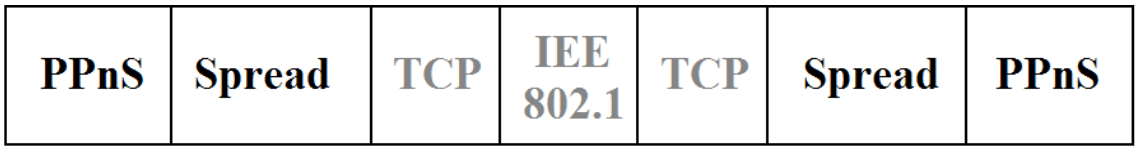
Prosessorin käyttöaste ilman viestiliikennettä mitataan seuraavasti. Käynnistetään käskyläinen, joka ajaa kymmentä PPnS:ää, kutakin omassa säikeessään. Top-ohjelma käynnistetään komennolla ”top -d 5”, jolloin prosessorin käyttöaste päivittyy viiden sekunnin välein. Tällöin lukemat ehditään kirjata ylös reaaliajassa. Lukemat kirjataan kahdestakymmenestä näytteestä ja niistä lasketaan lopuksi keskiarvo.

4.2.3 Viive

Tässä työssä viiveen määritellään tarkoittavan aikaa, joka viestillä kestää kulkea kahden tiedonsiirtoalustaa käyttävän ohjelman välillä. Tarkemmin määriteltynä sitä aikaväliä, kun viesti annetaan tiedonsiirtoalustalle ja se saadaan taas käyttöön vastinpäässä.

Tiedonsiirron viive koostuu monesta tekijästä. Tämän työn näkökulmasta viive voidaan kuitenkin jakaa kahteen osaan — aikaan, jonka paketti viettää tiedonsiirtoalustan kerroksissa ja aikaan, jolloin se on jossain muualla. Jako tehdään

siksi, että ohjelman ulkopuolisiin tekijöihin ei voida vaikuttaa ohjelmakoodia muuttamalla. Kuva 15 kuvaa työn kannalta oleelliset viiveen osat.



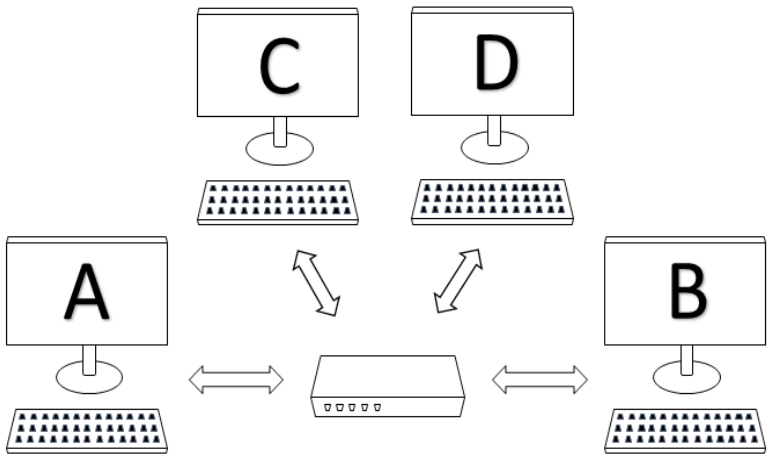
Kuva 15: Tiedonsiirron viiveen osa-alueet.

Viive, johon tämän työn osalta voidaan vaikuttaa rajoittuu Spread -tason yläpuolelle. PPnS-tasoa alempien kerrosten aiheuttama viive eliminoituu pois, kun testi ajetaan käyttäen kuormana Spread-tasoa.

Testin kulku on seuraavanlainen. Mittaaja lähettää viestin käskyläiselle, joka lähettää välittömästi vastausviestin. Tiedonsiirtoverkon voidaan olettaa olevan symmetrinen, eli viestin välittämiseen kuluu keskimäärin sama aika riippumatta siitä, kumpaan suuntaa viesti kulkee. Tästä johtuen viestien vaihtoon kuluva aika jaetaan kahdella, jolloin saadaan viestin välityksen viive.

4.3 Mittausympäristö

Mittaukset toteutettiin pääasiassa kohtalaisen tehokkailla pöytäkoneilla nopeassa LAN:ssa. Kuva 16 esittää käytettyä mittausympäristöä.



Kuva 16: Suorituskykytesteissä käytetyn mittausympäristön kaaviokuva.

Kuvassa 16 koneella A ajetaan mittaaja-ohjelmaa, koneella B käskyläisiä ja koneella C Spread Daemonia. Jako vastaa hyvin todellisuutta, sillä käskyläiset kuvaavat esimerkiksi sataman nostureita, mittaaja valvomon konetta ja spread daemonia ajetaan jossain erillisellä palvelimella. Koneella D ajetaan WANem:a (Wide Area Network emulator), mutta sitä käytetään vain kappaleessa 6.4 esitettyjen tulosten mittaamisessa. Koneet on yhdistetty samaan verkkoon nopealla 1Gbps kytkimellä. Verkko (LAN) on suljettu, eikä muilla laitteilla ole pääsyä siihen. Tämä takaa sen, että verkossa ei ole

muuta kuormitusta. Testikoneilla ei myöskään ole ajossa ylimääräisiä palveluita, joten niiden muu kuormitus on vähäinen. Koneiden keskeiset tekniset tiedot on koottu taulukkoon 2.

Taulukko 2: Testiympäristön koneiden tiedot

Kone	Suoritin	Muisti	Lisätiedot
A (mittaaja)	Intel core i7 @ 3,0 Ghz, 4 ydintä	4 Gt	
B (käskyläiset)	Intel core i7 @ 2,2 Ghz, käytössä 4 ydintä	Käytössä 4 Gt	Ajetaan virtuaalikoneessa (WMware)
C (daemon)	Pentium 4 @ 1,6 GHz	2 Gt	
D (WANem)	Pentium 4 @ 1,6 GHz	2 Gt	

Mittaaja-ohjelma on sijoitettu tehokkaimmalle tietokoneelle, koska sillä on suurin laskentatehovaatimus. Kevyet käskyläiset ajetaan toiseksi tehokkaimmalla koneella. Spread daemonia ja WANemia ajetaan vanhahkoilla pöytäkoneilla. Kaikkien koneiden käyttöjärjestelmänä toimi Ubuntu 12.04. Koneessa B isäntä käyttöjärjestelmänä toimii Windows 7. Virtuaalikoneen verkkoyhteys asetettiin siltaavaksi, eikä sen nopeutta rajoitettu ohjelmallisesti.

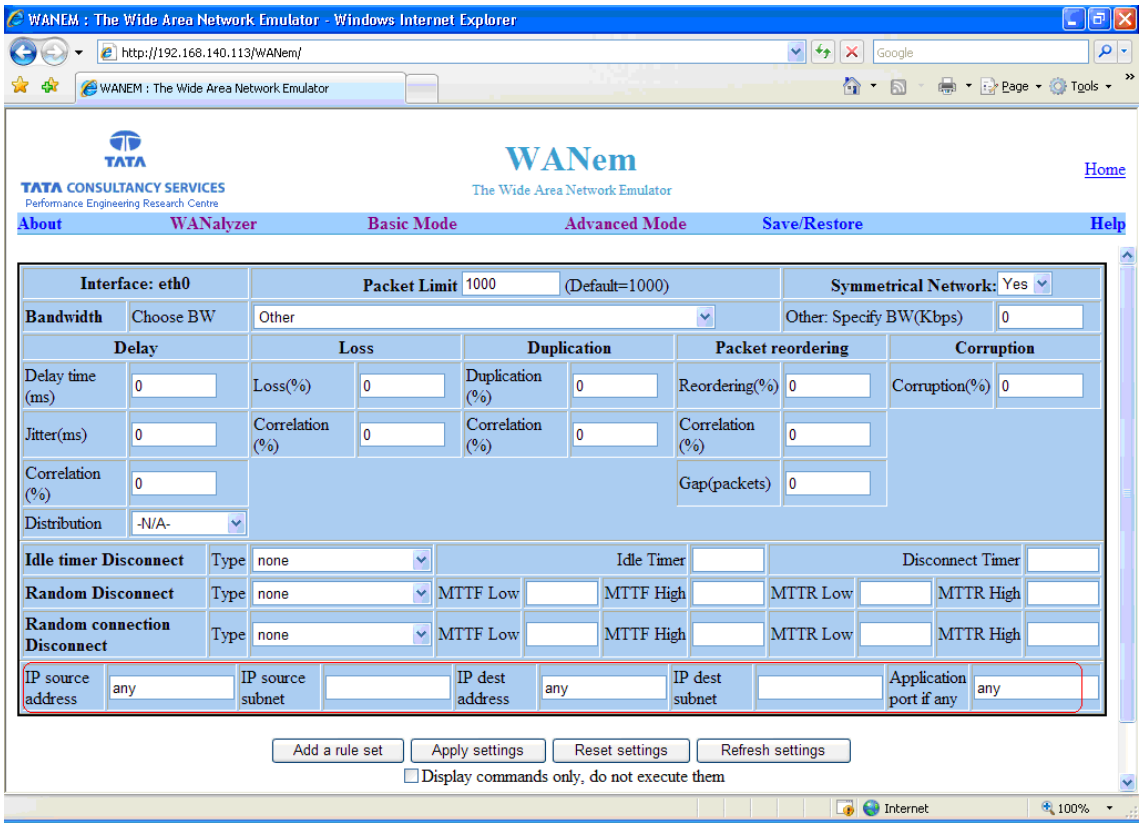
4.4 Mittauksen ja optimoinnin työkalut

Tässä kappaleessa käydään läpi mittauksessa ja optimoinnissa hyödynnetyt apuohjelmat. Mittauksissa käytettiin sekä valmiita ilmaisohjelmia kuin itsetehtyä mittausohjelmaa.

4.4.1 WANem

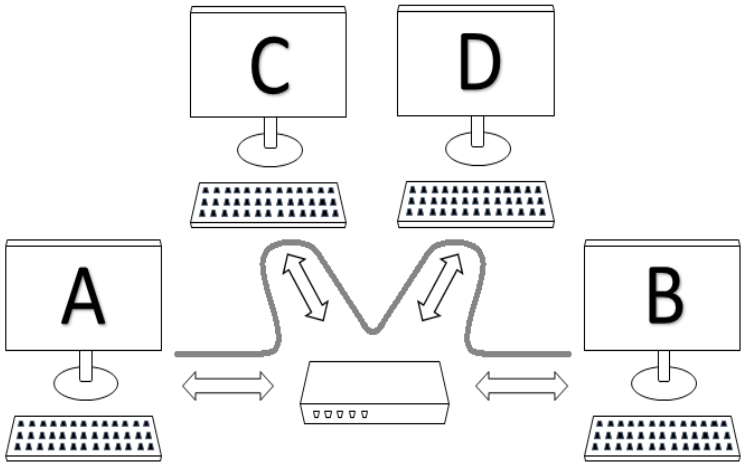
WANem (wide area network emulator) on laajaverkkojen (eng. wide area network, WAN) emulointiin tarkoitettu ohjelmisto. Tyypillisesti ohjelmistot kehitetään ja testataa lähiverkossa, joten WANem on erinomainen työkalu sellaisten ohjelmistojen kehityksessä, joissa yhteydet ovat internetin tai muun epäluotettavan verkon yli. WANem osaa simuloida muun muassa verkon viivettä, pakettihäviötä, paketin korruptoitumista ja yhteyden katkeamisia.[7]

Tässä työssä WANemia hyödynnettiin selvittämään, kuinka verkon viive ja pakettihäviö vaikuttavat tiedonsiirtoalustan tiedonsiirtokykyyn. Kuva 17 esittää WANemin käyttöliittymää.



Kuva 17: Esimerkkikuva WANem:n käyttöliittymästä [8].

Kuten kuvasta 17 voi päätellä, WANem:lla pystyy simuloimaan verkkoja hyvin monipuolisesti. Se asennetaan tavallisesti erilliselle tietokoneelle, joka kytketään yhden tai kahden verkkokortin avulla muuhun lähiverkkoon. Kahdesta verkkokortista on se hyöty, että yhden verkkokortin kapasiteettia ei tarvitse jakaa ulos ja sisään menevän liikenteen kesken. Työssä käytettiin kuitenkin yhtä verkkokorttia, koska toista korttia ei saatu toimimaan kyseisessä koneessa. Kuva 18 esittää käytettyä mittausympäristöä.



Kuva 18: Viiveen ja pakettihäviön vaikutuksen mittauksessa käytetyt laitteet ja niiden kytkennät. Harmaa viiva kuvaa viestien reittiä molempiin suuntiin

Kuvassa 18 *Tester* on mittaajan roolissa ja *Slave* käskyläisen roolissa. WANem:a ajetaan omalla tietokoneella. Kaikki neljä tietokonetta on kytketty samaan verkkoon kytkimen avulla. Reititystaulut muokattiin siten, että C- ja B-koneiden välinen liikenne ohjattiin D-koneen kautta. Tällöin mittajan ja daemonin väliin jää nopea sekä luotettava verkko. Daemonin ja käskyläisten väliin sitä vastoin jää WANem:lla simuloitava verkko. Kuvassa 18 näkyvä harmaa viiva kuvaa pakettien reittiä. Simuloitavan verkon ominaisuuksia muokattiin web-käyttöliittymän avulla ja mittaukset suoritettiin tarkoitusta varten kirjoitetulla ohjelmalla (kts. kappale 4.4.3).

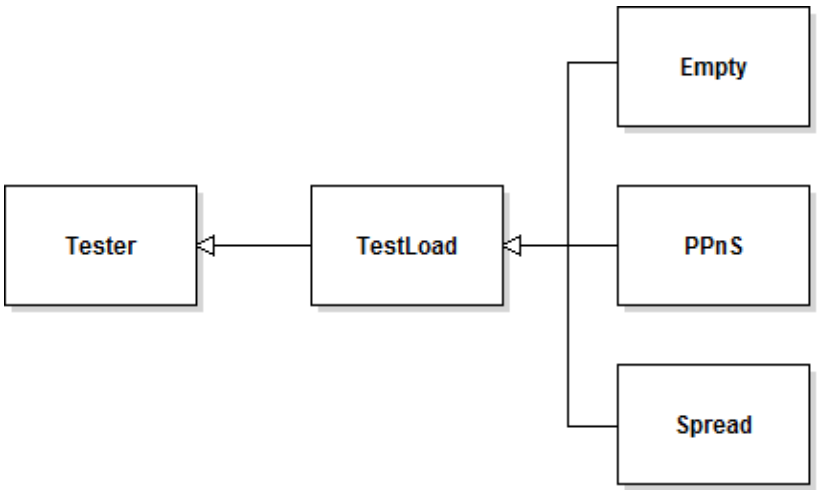
4.4.2 Valgrind

Valgrind on kokoelma hyödyllisiä ohjelmistokehityksessä ja -testauksessa käytettäviä työkaluja. Näiden työkalujen avulla voi tutkia tehokkaasti esimerkiksi muistivuotoja, moniajo-ongelmia ja ohjelman suorituskykyä. [9]

Tässä työssä käytettiin suorituskykyä analysoivaa työkalua. Lyhyesti kuvattuna ohjelma analysoi tutkittavan ohjelman funktiokutsuja, laskee niiden määrän sekä kunkin kutsun suorittamiseen käytetyn ajan. Analyysin tuottamasta listasta voi tutkia, viekö jokin operaatio tai funktiokutsu kohtuuttomasti aikaa.

4.4.3 Mittausohjelma

Mittausohjelmaan on aiemmin tässä luvussa viitattu nimellä mittaaja. Se on toteutettu käyttäen Qt:ta ja C++:aa. Ohjelma on kirjoitettu nimenomaan tiedonsiirtoalustan suorituskyvyn mittaamista varten. Mittausohjelma on toteutettu siten, että samat testit on helppo ajaa tiedonsiirtoalustan eri tasoille. Tällä tavalla pystytään erottelemaan kunkin tason tiedonsiirtokyvyn, prosessorikuorman ja viiveen. Kuva 19 esittää mittausohjelman yksinkertaistettua luokkakaaviota.



Kuva 19: Mittausohjelman ysinkertaistettu luokkakaavio.

Tester-luokassa on toteutettuna varsinaiset testit. TestLoad-luokka toteuttaa rajapinnan, jonka läpi voidaan käyttää kuormana tiedonsiirtoalustan eri tasoja. Käytettäessä kuormana PPnS-tasoa, kuormana on myös kaikki PPnS tasoa alemmat tiedonsiirtokerrokset. Vastaavasti Spread-tason tapauksessa, kuormana on kaikki sitä alemmat tiedonsiirtokerrokset. Empty-kuormalla voidaan mitata testiohjelman aiheuttama kuorma, mikä voidaan ottaa huomioon analysoitaessa muiden tasojen testituloksia.

4.5 Suorituskyky alussa

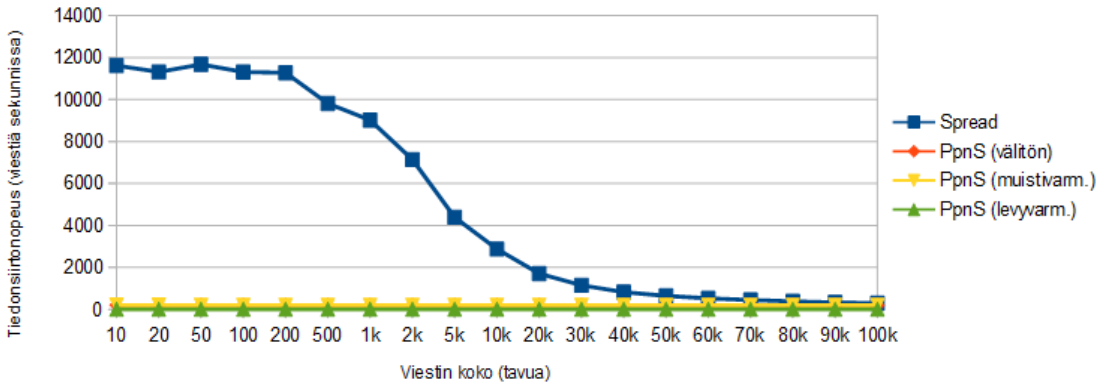
Tässä kappaleessa käydään läpi alustan suorituskyky työn aloitushetkellä. Tuolloin tiedonsiirtoalustan versio oli 2.3.1.1, joten tätä versiota käytetään vertailtaessa, kuinka paljon suorituskyvyn eri osa-alueet ovat muuttuneet työn edetessä. Alusta käännettiin Qt 4.8.5 vasten ja Spread Daemonista käytettiin versiota 4.3. Kunkin suorituskyvyn osa-alue on käyty läpi omassa kappaleessaan. Tulosten analysointi tehdään luvussa 5, koska siinä käydään läpi, mistä pullonkaulat johtuvat ja kuinka ne poistetaan.

4.5.1 Tiedonsiirtonopeus

Alustan tiedonsiirtonopeus mitattiin kappaleessa 4.2.1 kuvatuilla menetelmillä kappaleessa 4.3 kuvatussa ympäristössä. Kaikki testit ajettiin useaan kertaan ja lopuksi laskettiin keskiarvo. Testi, jossa molemmat osapuolet lähettävät viestejä yhtä paljon, ajettiin kahdella tavalla. Ensimmäisessä testissä käskyläisiä oli aina yksi kappale ja muuttuva tekijä oli viestin pituus. Toisessa testissä viestin pituus pidettiin vakiona (50 tavua) ja käskyläisten määrä muuttui.

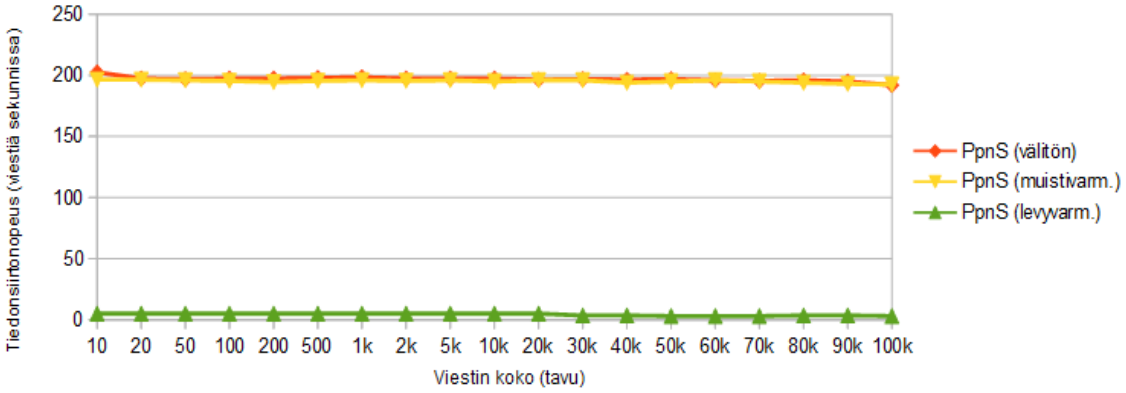
Jokaisen testin alussa luotiin yhteys mittaajan ja käskyläisen välille. Tämän jälkeen suoritettiin mittaus, eli lähetettiin viestit ja odottettiin vastaukset sekä mitattiin kulunut aika. Mittaus toistettiin viidesti ja jokaisen mittauksen välissä pidettiin sekunnin tauko. Lopuksi suljettiin yhteys ja siirryttiin seuraavaan viestinpituuteen tai lisättiin käskyläisten määrää riippuen testistä. Yhteensä mittauksia tuli jokaista arvoa kohden 25 kpl.

Testit ajettiin sekä Spread- että PPnS-tasoille. Kuvaan 20 on koottu molempien tasojen mittaustulokset yhden käskyläisen tapauksessa.



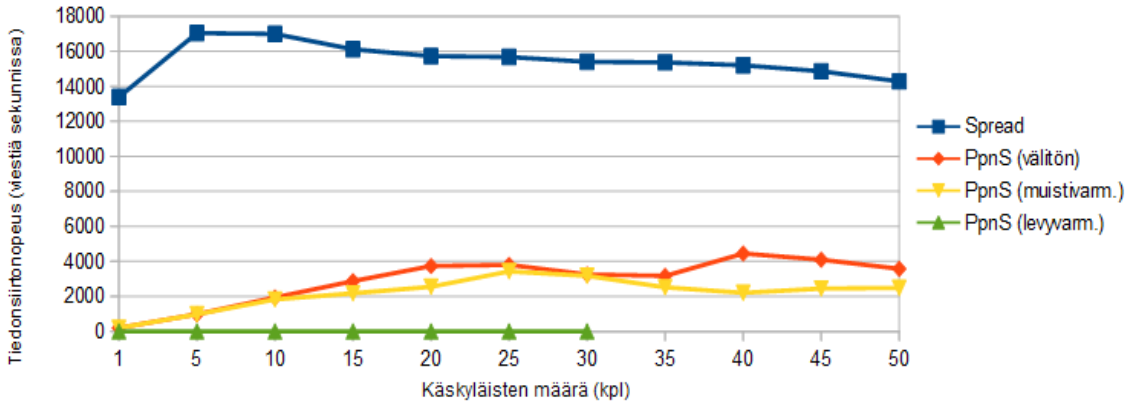
Kuva 20: Spread- ja PPnS-tason tiedonsiirtonopeudet viestin pituuden funktiona.

Kuvassa 20 on esitetty viestiparien lukumäärä, eli Spread Daemonin läpi kulkee kaksinkertainen määrä viestejä. Kuvasta näkee myös selvästi, että PPnS-taso on suuri pullonkaula tiedonsiirtoalustan suorituskyvyssä. Spread-taso kykenee välittämään parhaimmillaan noin 12 000 viestiä sekunnissa. PPnS-tason tiedonsiirtonopeus on kaikkien viestityyppien osalta niin alhainen, ettei tarkempia arvoja pysty erottamaan kuvaajasta. Samat tulokset on esitetty kuvassa 21 ilman spread-tason tulosta.



Kuva 21: PPnS-tason viestityyppien tiedonsiirtonopeudet viestin pituuden funktiona.

Viestin koko ei vaikuta tiedonsiirtonopeuteen välittömien ja muistivarmennettujen viestien tapauksessa. Levyvarmennetuissa viesteissä viestin pituudella on jonkin verran merkitystä, vaikka sitä ei kuvaajasta kovin helposti huomaa. Levyvarmennettuja viestejä voidaan lähettää pituudesta riippuen 5 – 10 kappaletta sekunnissa. Verkko ei myöskään ole pullonkaula, koska viestejä kulkee sama määrä, riippumatta niiden pituudesta. Kuvassa 22 on esitetty tulokset testeistä, joissa käskyläisten määrä muuttuu.



Kuva 22: Spread- ja PPnS-tasojen tiedonsiirtonopeudet käskyläisten määrän funktiona.

Kuvassa 22 on esitetty kaikkien käskyläisille lähetettyjen viestien summa. Ensimmäinen huomionarvoinen asia on se, että Spread Toolkit välittää viestejä nopeammin, jos asiakkaita on enemmän, kuin yksi.

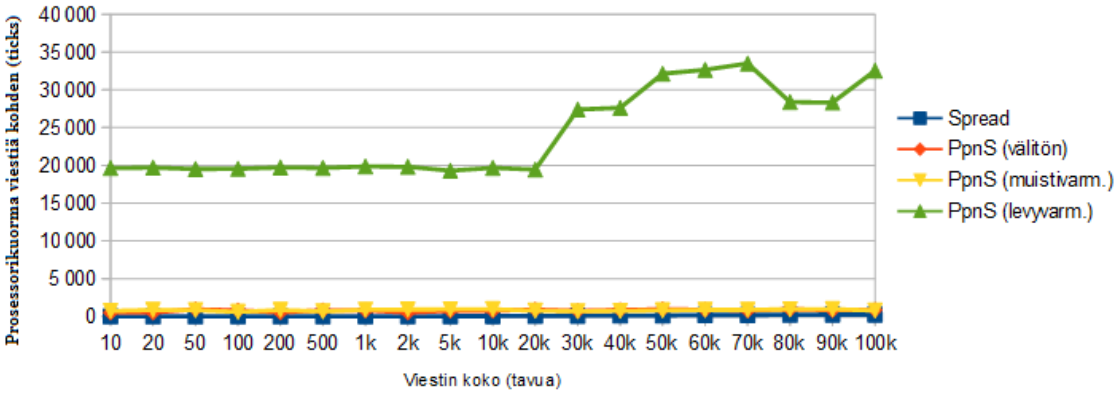
Välittömien viestien lukumäärä kasvaa lineaarisesti aina 20 käskyläiseen asti, kunnes käskyläisiä ajavan tietokoneen prosessorin suorituskyky tulee vastaan, sillä prosessori ei pysty käsittelemään viestejä nopeammin. Tämän jälkeen lähetysnopeus pysyy jotakuinkin vakiona, noin 4000 viestiä sekunnissa. Muistivarmennettujen viestien lähettäminen on hieman raskaampaa, joten niiden kohdalla prosessorin suorituskyky tulee vastaan aikaisemmin ja viestejä voidaan lähettää noin 3000 viestiä sekunnissa.

Levyvarmennettuja viestejä voidaan lähettää alle 10 viestiä sekunnissa riippuen käskyläisten määrästä. Kun käskyläisiä oli yli kolmekymmentä, viestien lähetysnopeus romahti niin alhaiseksi, että testit jätettiin ajamatta suuremmilla määrilllä.

Tässä työssä ei esitetä muita alkuperäiselle versiolle ajettuja testejä, koska ne eivät tuo oleellista lisäinformaatiota. Todetaan vain, että suurin pullonkaula tässä vaiheessa on PPnS-taso.

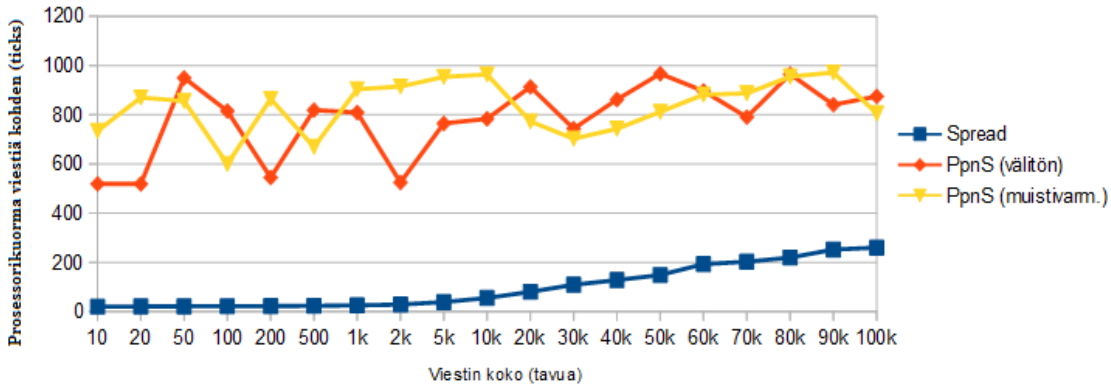
4.5.2 Prosessorikuorma

Alustan prosessorin käyttöaste mitattiin kappaleessa 4.2.2 kuvatulla menetelmällä. Testit ajettiin useaan kertaan ja niistä laskettiin lopuksi keskiarvo. Prosessorikuorman mittaamiseksi suoritettiin samat testit, kuin tiedonsiirtonopeudelle, sillä kyseessä on sama testikokoelma. Samalla testillä voidaan mitata sekä tiedonsiirtonopeus että prosessorikuorma. Prosessorikuorma on mitattu A-koneesta. Siinä on mukana viestien lähettämisen ja vatsaanottamisen yhteinen prosessoriaika. Kuvassa 23 on esitetty saadut tulokset yhden käskyläisen tapauksessa. Tulokset on laskettu kaavan 4 avulla, eli niistä on poistettu testiohjelman aiheuttama prosessorikuorma.



Kuva 23: Spread- ja PPnS-tasojen prosessorinkäyttö viestin pituuden funktiona.

Kuvasta 23 voidaan nähdä lähinnä se, että levyvarmennettujen viestien välittäminen on äärimmäisen raskasta verrattuna muihin viestityyppeihin. Tästä syystä kuvassa 24 on esitetty samat tulokset ilman levyvarmennettuja viestejä.



Kuva 24: Spread- ja PPnS-tasojen prosessorinkäyttö viestin pituuden funktiona.

Kuvasta näkee, että Spread Toolkitin käyttämä prosessoriaika on todella pieni verrattuna PPnS:n välittömien ja muistivarmennettujen viestien käyttämään prosessoriaikaan. PPnS:n viestien aiheuttamassa prosessorikuormassa on melko suuria vaihteluita. Vaihtelua saataisiin varmasti pienennettyä ajamalla testejä selvästi enemmän ja laskemalla niistä keskiarvo. Kuvaajasta voidaan kuitenkin lukea, että vaadittu prosessorikuorma on pienillä paketeilla noin 700 ja suurilla paketeilla noin 800.

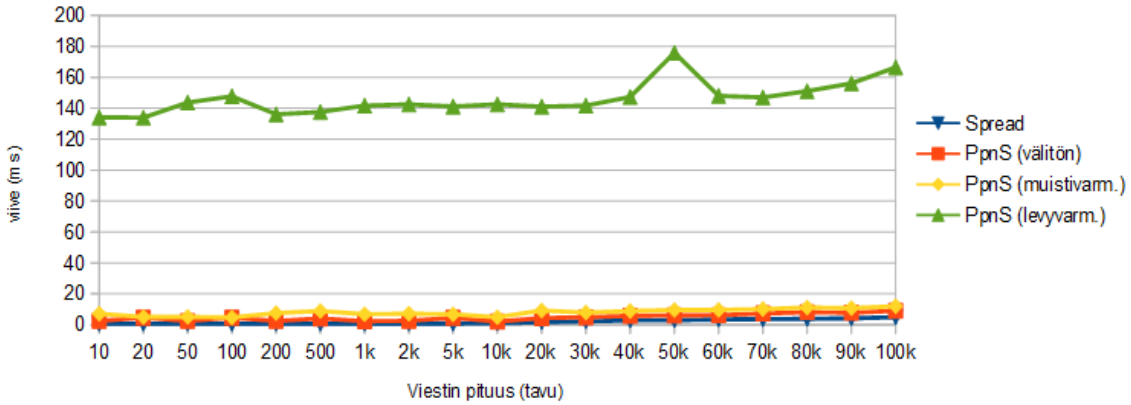
Alkuperäiselle versiolle ajettiin kappaleessa 4.2.2 esitetty tyhjäkäyntikuorman mittaus. Mittaus suoritettiin samalla koneella, kuin käskyläisiä ajetaan muissakin testeissä. Koneen tiedot on esitetty taulukossa 2 (kone A). Tulokseksi saatiin 14,9 % / käskyläinen. Tulos on todella suuri luku ohjelmalle, joka ei tee muuta, kuin ylläpitää yhteyttä.

4.5.3 Viive

Alustan aiheuttama tiedonsiirtoviive mitattiin kapaleessa 4.2.3 kuvatulla menetelmällä. Testit ajettiin useaan kertaan ja niistä laskettiin lopuksi keskiarvo.

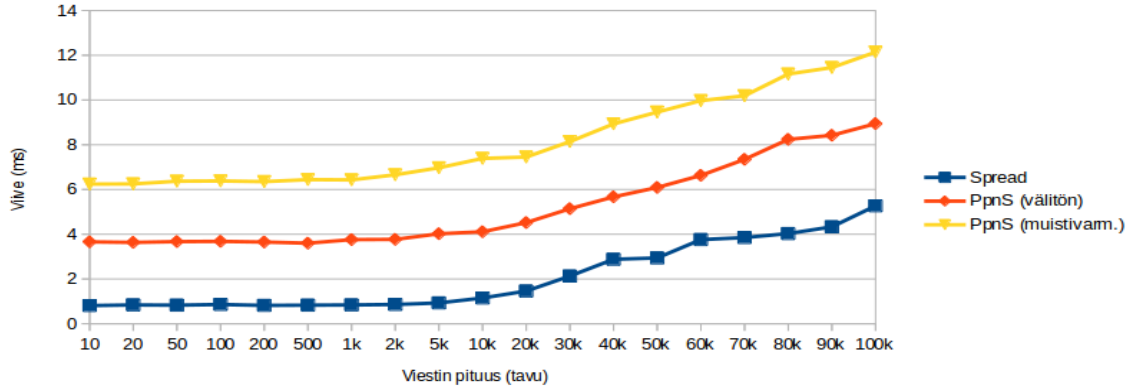
Tiedonsiirtoalustan alkuperäisestä versiosta mitattiin vain PPnS-tason aiheuttama viive. Tämä toteutettiin ajamalla testit sekä Spread-tasolle että PPnS-tasolle ja laskemalla lopuksi näiden erotus. Viestejä lähetettiin puolen sekunnin välein, jotta tiedonsiirtoalustan kuormitus ei vääristäisi tuloksia. Jos viestejä lähetettäisiin liian nopeasti, osa viesteistä voisi joutua odottamaan jonossa edellisiä viestejä, jolloin niiden viive olisi suurempi.

Viivetestejä ajettiin sekä yhdelle että useammalle käskyläiselle. Yhden käskyläisen tapauksessa muutettiin viestin pituutta. Usean käskyläisen tapauksessa viestin pituus pidettiin vakiona ja muuttuvana tekijänä pidettiin käskyläisten määrää. Yhden käskyläisen tapauksessa saadut tulokset on koottu kuvaan 25.



Kuva 25: Spread- ja PPnS-tasojen tiedonsiirtoviiveet.

Kuvasta näkee, että levyvarmennettujen viestien viive on omaa luokkaansa. Sen viiveet vaihtelevat 130 ja 180 ms:n välillä, kun muiden viiveet ovat lähellä nollaa. Tästä syystä samat tulokset on esitetty kuvassa 26 ilman levyvarmennettuja viestejä.



Kuva 26: Spread- ja PPnS-tasojen tiedonsiirtoviiveet (ilman levyvarmennettua viestiä).

Viiveet kasvavat portaittain siten, että Spread-tason viive on alhaisin, välittömillä viesteillä kestää hieman pidempään ja muistiivarmennetuilla viesteillä vielä pidempään.

5 TIEDONSIIRRON OPTIMOINTI

Tässä luvussa käydään läpi ohjelmistoon tehdyt muutokset, joiden seurauksena tiedonsiirtonopeus kasvoi, prosessorikuorma pieneni tai viestin välityksen viive lyhenin. Kaikki pullonkaulat löytyivät koodikatselmoinnilla ja niiden vaikutus suorituskykyyn todennettiin luvussa 4 kuvatuilla mittausmenetelmillä. Yksittäisten muutosten vaikutusta tiedonsiirtokykyyn tai prosessorikuoman alenemiseen ei esitetä tässä työssä tarkasti vaan tyydytään toteamaan, mihin asioihin muutos vaikuttaa ja onko muutos merkittävä vai marginaalinen. Kaikkien muutosten yhteisvaikutus on esitetty seuraavassa luvussa 6.

Ongelmista ja niiden ratkaisuista on esitetty keskeiset ohjelmakoodit. Suurimmat pullonkaulat löytyivät viestien vastaanottamisesta, vuorontajan toiminnasta ja tietokannan käytöstä. Nämä osa-alueet on käsitelty omissa alaluvuissaan.

5.1 Viestien vastaanottaminen

Viestien vastaanottaminen on kuvattu ajoituskaavion avulla kappaleessa 3.3. Tässä kappaleessa käydään läpi viestien vastaanottaminen kooditasolla. Se on ainoa tapa, jolla optimoinnin yhteydessä tehdyt muutokset voidaan esittää lukijalle.

Kuten kappaleessa 3.3 todettiin, viestejä luetaan viiden millisekunnin välein. Viestejä ei ole hyvä lukea niin nopeasti, kuin mahdollista, sillä se kuormittaisi prosessoria turhaan. Tästä kuitenkin aiheutuu suuri pullonkaula, jos viestejä luetaan kerralla vain yksi kappale. Alla on esitetty alkuperäinen viestin vastaanottaminen kooditasolla oleellisilta osin.

```
void Receiver::timerEvent(QTimerEvent * event) {
    int ret = SP_poll(m_mailBox);
    if (ret <= 0) {…} // Virheenkäsittely
    ret = SP_receive(m_mailBox, message, …);
    if (ret < 0) {…} // Virheenkäsittely
    emit message;
}
```

Receiver-luokka sisältää QBasicTimer-tyyppisen olion. Kyseessä on Qt-kirjaston ajastin, jolla voidaan tuottaa signaaleita halutuin väliajoin. Kun Receiver-luokan instanssi alustetaan, ajastin asetetaan tuottamaan signaaleita viiden millisekunnin välein. Ajastimen tuottamat signaalit käsitellään yllä kuvatussa timerEvent-metodissa.

Rivillä kaksi kutsutaan Spread Toolkit:n asiakas kirjaston (Spread client) *SP_poll*-funktiota. Funktion paluuarvo kertoo, onko viestejä luettavissa. Mahdolliset virheet

käsitellään rivillä kolme. Jos uusi viesti on luettavissa, se luetaan rivillä neljä. *SP_receive*-funktion paluuarvo kertoo, onnistuiko viestin lukeminen ja viestin hyötykuorma tallentuu message-muuttujaan. Rivillä viisi tarkastetaan mahdolliset virheet ja rivillä kuusi välitetään vastaanotettu viesti eteenpäin.

Menetelmässä on kaksi isoa ongelmaa. PPnS:aa ajettiin Valgrind-ohjelmalla ja havaittiin, että *SP_poll*-funktio on melko raskas operaatio. Tätä funktiota kutsutaan 200 kertaa sekunnissa riippumatta siitä vastaanotetaanko viestejä vai ei. Lisäksi jokaista ajastimen tuottamaa signaalia kohden luetaan vain yksi viesti, joten viestejä voidaan vastaanottaa maksimissaan 200 sekuntia kohden. Tämä selittää kappaleessa 4.5.1 havaitun suuren eron Spread- ja PPnS-tasojen välisessä tiedonsiirtonopeudessa.

Viestien vastaanottamisen rajoitus voidaan poistaa monella tavalla. Helpoin tapa olisi lisätä koodiin muutama rivi koodia, jossa jokaisen onnistuneen luvun jälkeen pyörittäisiin while-silmukassa lukien viestejä, kunnes *SP_poll*-funktio ilmoittaa viestien loppuneen. Tässä on se ongelma, että viestien lukeminen voi kestää huomattavasti kauemmin, kuin viisi millisekuntia, jolloin signaaleita kertyy jonoon turhaan. Lisäksi täytyy muistaa, että Valgrind osoitti *SP_poll* funktion olevan raskas operaatio.

Viestien vastaanottaminen päätettiin kirjoittaa lopulta kokonaan uusiksi. *SP_poll*-funktio päätettiin jättää pois ja tilalle valittiin *select*-funktio, joka löytyy melkein samanlaisena sekä Windows että Linux käyttöjärjestelmistä. Tämä on mahdollista, koska Spread:n rajapinta tarjoaa yhteyden tiedostokahvan (eng. file descriptor). Myös ajastimen käytöstä luovuttiin ja luku siirrettiin while-silmukkaan. Alla on esitetty lopullinen viestien vastaanottamisesta huolehtiva koodi niiltä osin, kuin se on tarpeellista.

```
void Receive::start() {
    while (m_run) {
        ret = select(m_mailbox, timeout, …);
        if (ret == 0) continue;
        if (ret < 0) {…} // Virheenkäsittely
        ret = SP_receive(m_mailbox, message, …);
        if (ret < 0) {…} // Virheenkäsittely
        emit message;
    }
}
```

Rivillä kaksi määritellään while-silmukka. Silmukasta voidaan voidaan poistua hallitusti asettamalla m_run epätodeksi.

Rivillä kolme kutsutaan *select*-funktiota. *Select*-funktio jää odottamaan, kunnes uusi viesti on luettavissa tai suurin sallittu odotusaika on kulunut (timeout). Maksimiodotusaika valittiin kokeilemalla ja päädyttiin lopulta arvoon 200 ms. Tällöin funktiota ei kutsuta turhan usein, joten se ei aiheuta havaittavaa tyhjäkäyntikuormaa

(eng. idle load). 200 ms on edelleen lyhyt aika, joten silmukasta päästään nopeasti pois, vaikka viestejä ei saapuisikaan.

Jos viestiä ei vastaanoteta 200 ms:n aikana, *select*-funktio palauttaa nollan, jolloin rivin neljä ehtolause toteutuu ja palataan silmukan alkuun. Viestin saapuessa *select*-funktio palauttaa välittömästi viestin pituuden, jonka jälkeen viesti luetaan SP_receive-funktiolla rivillä kuusi. Jos viestin luku onnistuu, se välitetään eteenpäin rivillä kahdeksan.

Lopputuloksena syntyi kevyt ja nopea tapa vastaanottaa saapuvat viestit. Muutos alensi prosessorikuormaa selvästi ja poisti vastaanottamisesta turhan rajoituksen (200 viestiä sekunnissa).

5.2 Vuorontaja

Kuten edellä on todettu, PPnS tukee useita eritasoisia takuita viestien välittämiseen. Kolme neljästä viestityypistä käyttää viestipuskureita. Viestipuskureiden kautta kulkevien viestien lähettämisestä päättää vuorontaja, joten sen toiminnan nopeus ja tehokkuus on ratkaisevassa roolissa tiedonsiirrossa. Viestipuskurien käyttö lähettämisessä on selitetty tarkemmin edellä kappaleessa 3.2.

Alkuperäisessä versiossa vuorontaja toimii siten, että se käy viiden millisekunnin välein läpi kaikki Connection-luokan instanssiin liittyvät PeerChannel-luokan instanssit ja lähettää jokaisesta PeerChannel-luokan instanssista korkeintaan yhden viestin kierrosta kohden. Kuvassa 27 on esitetty vuorontajaan liittyvä ohjelmakoodi oleellisilta osin.

```
1:void Connection::send() {
2:    foreach (AbstractChannel* channel, m_channelList) {
3:       if (channel->schedule(message)) {
4:          multicast(message);
5:       }
6:    }
7:}
8:
9:void PeerChannel::schedule(Message& msg) {
10:   bool messageFound = false;
11:   foreach (AbstractQueue* queue, m_queues) {
12:      if (!queue->isEmpty()) {
13:         queue->head(msg);
14:         messageFound = true;
15:         break;
16:      }
17:   }
18:   return messageFound;
19:}
```

Kuva 27: Alkuperäisen vuorontajan koodi oleellisilta osin.

Connection-luokka sisältää QBasicTimer tyyppisen ajastimen, joka tahdistaa vuorontajan toiminnan. Ajastin tuottaa signaaleita viiden millisekunnin välein samalla tavalla, kuin viestien vastaanottamisessa. Signaalit käsitellään Connection-luokan *send*-metodissa. Rivillä kaksi on foreach-lause, jolla käydään läpi kaikki Connection-luokan instanssiin liittyvät kanavat (AbstractChannel). Rivillä kolme kutsutaan käsiteltävän kanavan *schedule*-metodia, joka hakee kanavalta seuraavaksi lähetettävän viestin. *Schedule*-metodi on esitelty riviltä yhdeksän eteenpäin.

Schedule-metodissa on rivillä 11 foreach-lause, jolla käydään läpi kaikki kanavan viestijonot. Viestijonot käydään läpi prioriteettijärjesteyksessä, kunnes jostain jonosta (queue) löytyy lähetettävä viesti. Viesti haetaan kutsumalla jonon *head*-metodia, joka palauttaa jonon ensimmäisen viestin msg-muuttujaan (rivi 13). *MessageFound*-muuttujan arvo merkataan todeksi ja poistutaan foreach-lauseesta break-lauseella. Lopuksi palautetaan tieto siitä, löytyikö kanavalta lähetettävä viesti (rivi 18). Viesti lähetetään tarvittaessa rivillä neljä kutsumalla Connection-luokan *multicast*-metodia.

Vuorontajan toteutuksessa on samanlainen ongelma, kuin viestien vastaanottamisessa. Se rajoittaa lähetettävien viestien määrää. Maksiminopeus on 200 viestiä sekunnissa jokaista kanavaa kohden. Viestimäärän rajoittamisen lisäksi vuorontaja tekee paljon turhaa työtä erityisesti silloin, kun viestejä ei liiku. Se käy kaikkien kanavien viestijonot läpi 200 kertaa sekunnissa.

Tämäkin ongelma voidaan ratkaista monella tavalla. Tässä työssä esitetään kaksi ratkaisua: toteutettu ratkaisu sekä myöhemmin visioitu, selvästi parempi ratkaisu. Näistä jälkimmäinen käydään läpi kappaleessa 7.1.

Työn aikana toteutetussa ratkaisussa ideana on vuorontajan tarkastusintervallin muuttaminen tilanteen mukaan. Optimaalinen tilanne olisi se, että vuorontaja pysäytetään, kun yhdelläkään kanavalla ei ole viestejä ja käynnistetään, kun johonkin jonoon lisätään viesti. Vuorontaja voisi lähettää jonoissa olevia viestejä tauotta, kunnes ne loppuvat.

Käytännön toteutus on tehty melkein ylläkuvatulla tavalla, mutta siinä on tehty yksi poikkeus käytännön syistä. Edellä on mainittu että, muutokset on tehty aina suoraan tuotantokoodii. PPnS-tason on melko monimutkainen kokonaisuus ja se sisältää paljon erikoistilanteiden käsittelyä. Tästä johtuen vuorontajaa ei koskaan pysäytetä kokonaan vaan se hidastetaan niin, että pysähtyneenä se tarkastaa jonot kerran sekunnissa. Tällä halutaan varmista, että vuorontaja ei jätä käsittelemättä mitään viestiä, vaikka se jäisikin käynnistämättä jonkin erikoistilanteen yhteydessä. Käytännön toteutuksen oleelliset kohdat on esitetty kuvassa 28.

```
1: void Connection::send() {
2:    bool messageSent = false;
3:    m_runTimer.start(SLOW_INTERVAL);
4:    foreach (QAbstractChannel* channel, m_channelList) {
5:       if (channel->schedule(message)) {
6:          multicast(message);
7:          messageSent = true;
8:       }
9:    }
10:
11:   QMutexLocker locker(&m_lock);
12:   if (messageSent || m_fastSending) {
13:      m_runTimer.start(0);
14:      m_fastSending = messageSent;
15:   }
18:}
19:
20:void Connection::speedUpScheduler() {
21:   QMutexLocker locker(&m_lock);
22:   if (!m_fastSending) {
23:      m_runTimer.start(0);
24:   }
25:}
```

Kuva 28: Uuden vuorontajan ohjelmakoodi oleellisilta osin

Connection-luokka sisältää QTimer-tyyppisen ajastimen (koodissa m_runTimer), joka tahdistaa vuorontajaa. Ajastimen tuottamat signaalit käsitellään Connection-luokan send-metodissa (kuva 28). Alussa oletetaan, että viestejä ei tulla lähettämään, joten messageSent muuttuja alustetaan epätodeksi. Samasta syystä m_runTimer:n intervalli asetetaan yhteen sekuntiin (SLOW_INTERVAL). Tällöin ajastin ei myöskään tuota signaaleja sillä välin, kun muita viestejä lähetetään. Tämän jälkeen käydään läpi kanavien viestijonot ja lähetetään viestit samalla tavalla, kuin alkuperäisessäkin toteutuksessa. Jos lähetetään yksikin viesti, messageSent asetetaan todeksi (rivi 7).

Sitten tutkitaan, jatketaanko viestijonojen tarkastelua nopealla tahdilla vai hidastetaanko tahtia. Jos viestijonoista löytyi lähetettävä viesti (messageSent on tosi) tai nopean lähetyksen lippu on päällä (m_fastSending), ajastimen intervalli asetetaan nollaksi, jolloin vuorontaja tutkii viestijonoja mahdollisimman taajaan (rivi 13). Rivillä 14 nopean lähetyksen lipulle asetetaan uusi arvo sen mukaan, lähetettiinkö viestiä vai ei. Tästä seuraa, että viestijonojen tyhjenemisen jälkeen jonot käydään tarkastamassa kertaalleen.

5.3 Tietokanta

Levyvarmennetut viestit kierrätetään massamuistin kautta, jotta ne eivät hukkuisi, vaikka laitteen virrat katkeaisivat. Viestit tallennetaan levylle käyttäen SQLite-tietokantaa. SQLite on kevyt c:llä toteutettu tietokantajärjestelmä, joka linkitetään

sovellusohjelmaan [10]. Erillistä tietokantaserveriä ei ole. SQLite on toteutettu Qt4:ssa liitännäisen avulla.

Alkuperäisessä PPnS:n toteutuksessa SQLite kantaa käytettiin oletusasetuksilla. Oletusasetukset eivät ole optimaaliset PPnS:n käyttötarkoitukseen, joten optimointivaraa löytyi runsaasti. Toteutetut optimoinnit on käyty läpi omissa alaluvuissaan.

5.3.1 Tietokannan synkronointitapa

Perinteisten tietokantojen yksi keskeisimmistä ominaisuuksista on se, että tieto on aina ehjää. Tämä tarkoittaa sitä, että kirjoitusoperaatiot tehdään aina kokonaisuudessaan, jolloin luettaessa osa tiedosta ei voi olla vanhaa ja osa uutta. Lisäksi varmistetaan, että tieto kirjoitettiin kantaan oikein ja kokoinaisuudessaan sellaisissakin tilanteissa, joissa laitteesta katkeaa virta kesken kirjoituksen.

Synkronoinnin haittapuolena on sen negatiivinen vaikutus suorituskykyyn. SQLite tarjoaakin neljä eri tasoa synkronointiin, joten käyttäjä voi tehdä kompromisseja tietokannan eheyden ja suorituskyvyn välillä. Nämä tasot ovat: NONE, NORMAL, FULL ja EXTRA. SQLiten oletus synkronointitapa on FULL. FULL-synkronointitapa varmistaa, että järjestelmän kaatuminen tai virtojen katkeaminen ei korruptoi tietokantaa. FULL-synronointitapa on kuitenkin hyvin hidas verrattuna NORMAL-synkronointitapaan.[11]

NORMAL-synkronointitavan käyttö nosti PPnS:n levyvarmennettujen viestien välitysnopeuden noin satakertaiseksi. Haittapuolena on se, että tietokannan eheyttä ei voida taata yllättävän virtakatkon yhteydessä. Dokumentaatio kuitenkin huomauttaa, että korruptoitumisen todennäköisyys on pienempi, kuin kiintolevyn hajoamisen todennäköisyys [11]. Riski on nopeushyödyn arvoinen, joten synkronointitavaksi valittiin NORMAL.

NONE-synkronointitapa olisi kaikista nopein, mutta se ei tee minkäänlaisia tarkistuksia. Viestien tulee säilyä yllättävien virtakatkojen yli, vaikka kyse ei olekaan ihmishengistä.

5.3.2 Tietokannan käsittelytapa

SQLite tarjoaa kaksi erilaista tapaa käsitellä tietoantaa: Palautusmuistio (eng. rollback journal) ja kirjoitusloki (eng. write-ahead log). Palautusmuistio on perinteinen tapa käsitellä tietokantaa. Kirjoitusloki on uudempi tapa, mutta se asettaa järjestelmälle tiettyjä rajoitteita.

Palautusmuistio toimii seuraavasti. Kirjoitettaessa tietokantaan, tietokantatiedostosta kopioidaan talteen väliaikaistiedostoon ne osat, joihin kirjoitusoperaatio tekee muutoksia. Tämän jälkeen tehdään tarvittavat muutokset suoraan tietokantatiedostoon. Jos kirjoittaminen onnistuu, väliaikaistiedosto voidaan poistaa. Jos jokin menee pieleen,

tietokanta palautetaan vanhaan muotoonsa käyttämällä väliaikaistiedostoon tallennettuja tietoja.[12]

Kirjoituslokin tapauksessa tiedoista ei oteta kopioita kirjoitushetkellä vaan uudet tiedot kirjoitetaan eräänlaiseen väliaikaistiedostoon. Tämä väliaikaistiedosto sisältää tiedot viimeisimmistä tietokantaan tehdyistä muutoksista ja sen sisältö päivitetään varsinaiseen tietokantatiedostoon, kun sen koko kasvaa riittävän suureksi. Lukuoperaatioissa tarkastetaan ensin väliaikaistiedoston sisältö, jonka jälkeen luetaan tarvittaessa varsinaisen tietokantatiedoston sisältö. Kirjoituslokista saadaan merkittävää nopeushyöty, jos tietokantaan tehdään paljon pieniä muutoksia. Kirjoitusloki vaatii toimiakseen järjestelmältä jaetun muistin (eng. shared memory) tuken. Toinen vaihtoehto on rajoittaa yhtäaikaisten tietokantayhteyksien määrä yhteen.[13]

Työssä päädyttiin jälkimmäiseen vaihtoehtoon, koska kaikki tuetut alustat eivät tarjonneet jaetun muistin tukea ja yhtäaikaisia tietokantayhteyksiä tarvitaan vain yksi.

5.3.3 Tietokantakutsujen valmistelu

Tietokantakutsut on hyvä valmistella (eng. prepare) ennen niiden suorittamista. Valmistelu nopeuttaa kutsujen suorittamista sellaisissa tapauksissa, joissa samanlaista kutsua toistetaan useasti. Tämä tapahtuu niin sanotun kyselyn suoritussuunnitelman avulla (eng. query plan). Suunnitelma tehdään ensimmäisellä kutsukerralla ja sitä hyödynnetään myöhemmillä kerroilla.[14]

Tietokantakutsuja ei valmisteltu alkuperäisessä versiossa, joten kutsut muutettiin valmistelluiksi. Tietokantakutsujen valmistelu tehtiin viimeisenä ja se paransi viestien välitysnopeutta noin 20%.

5.4 Muut parannukset

Ylläkuvattujen optimointien lisäksi PPnS:stä löytyi muutama kohta, jossa tietoa kopiotiin turhaan funktiosta toiseen. Turhat kopioinnit muutettiin viittauksiksi, minkä pitäisi parantaa suorituskykyä ainakin teoriassa. Näiden parannusten vaikutus suorituskykyyn on joka tapauksessa marginaalinen verrattuna muihin parannuksiin.

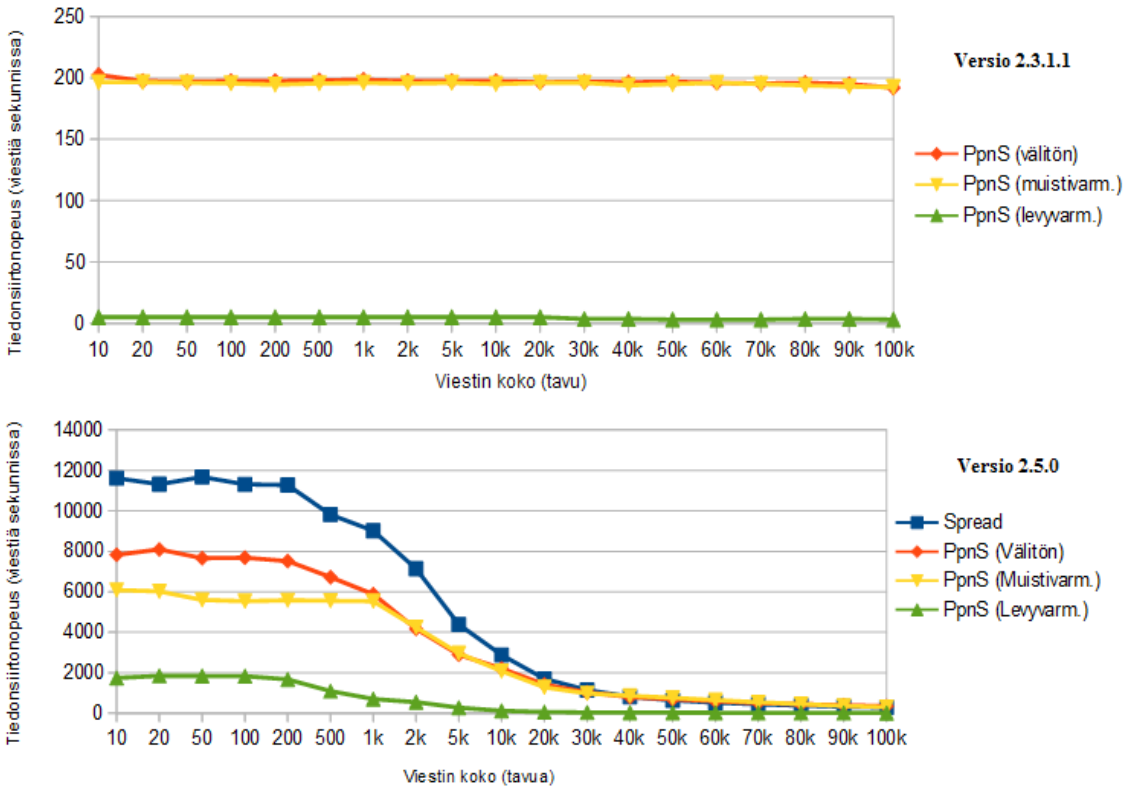
Suorituskykymittaukset asettivat alustan myös ennennäkemättömälle rasitukselle, mikä paljasti muutamia epävakausongelmia. Nämä saatiin korjattua ja testattua mittausten avulla.

6 TULOKSET

Tässä luvussa käydään läpi työn suorituksen aikana syntyneet parannukset tiedonsiirtokyvyn kasvun, prosessorikuorman vähenemisen sekä tiedonsiirtoviiveen pienenemisen osalta. Tiedonsiirtoalustalle tehtiin kaikki luvussa 5 kuvatut muutokset. Luvussa käsitellään lyhyesti myös alempien tiedonsiirtokerrosten asettamat rajoitteet ja niiden vaikutus suorituskykyyn.

6.1 Tietonsiirtokyky

Tiedonsiirron osalta saavutettiin merkittäviä tuloksia, sillä nopeudet kasvoivat moninkertaisiksi. Suorituskykymittauksia ajettiin muutostöiden aikana, jotta voitiin osoittaa, että ne parantavat suorituskykyä. Tässä kappaleessa käydään läpi mittaustulokset, jotka on ajettu kaikkien toteutettujen muutosten jälkeen alustan versiolle 2.5.0. Tiedonsiirtokyvyn mittaamiseksi ajettiin kaikki kappaleessa 4.2.1 kuvatut mittaukset. Kuvassa 29 on mittaustulokset testistä, jossa viestejä liikkuu molempiin suuntiin yhtä paljon. Käskyläisiä on yksi ja muuttuvana tekijänä on viestin koko. Ylemmässä kuvaajassa on vanhan version tulokset vertailun vuoksi (kuva 21).

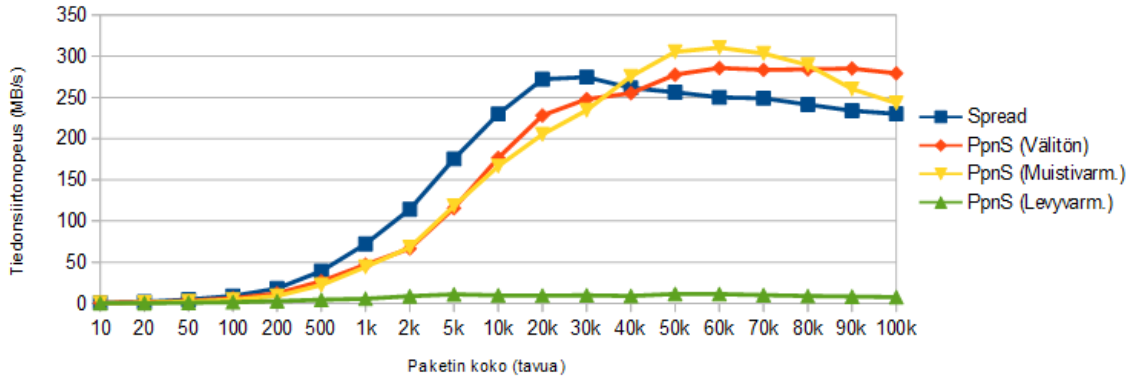


Kuva 29: Spread- ja PPnS-tasojen tiedonsiirtonopeudet viestin pituuden funktiona.

Kuvassa 29 on ilmoitettu viestiparien lukumäärä. Toisin sanoen Spread Daemonin läpi kulkee kaksinkertainen viestimäärä kuvaajan arvoon nähden. Pelkällä Spread Toolkitilla ajettuna viestejä saadaan kulkemaan selvästi eniten (noin 12000 viestiä sekunnissa). Seuraavaksi eniten kulkeen PPnS:n välittömiä viestejä (noin 8000 viestiä sekunnissa), kolmanneksi nopeiten PPnS:n muistivarmennettuja viestejä (noin 6000 viestiä sekunnissa) ja hitaiten kulkevat levyvarmennetut viestit (noin 2000 viestiä sekunnissa). Käytännön sovelluksissa iso osa viesteistä on alle 200 tavun mittaisia ja suurin osa alle kilotavun mittaisia.

Viestien nopeudet ovat siinä järjestyksessä, kuin voisi olettaakin, raskauden mukaan. Yksi poikkeus kuitenkin löytyy, sillä yhtä kilotavua suuremmat välittömät ja muistivarmennetut paketit näyttävät olevan yhtä nopeita lähettää. Tämän ilmiön selitys piillee selvästi pintaa syvemmällä. Mahdollisia syitä voivat olla viestien pilkkoutuminen alemmilla tasoilla (esim. TCP) tai erilaiset puskurit (esim. verkkokortti).

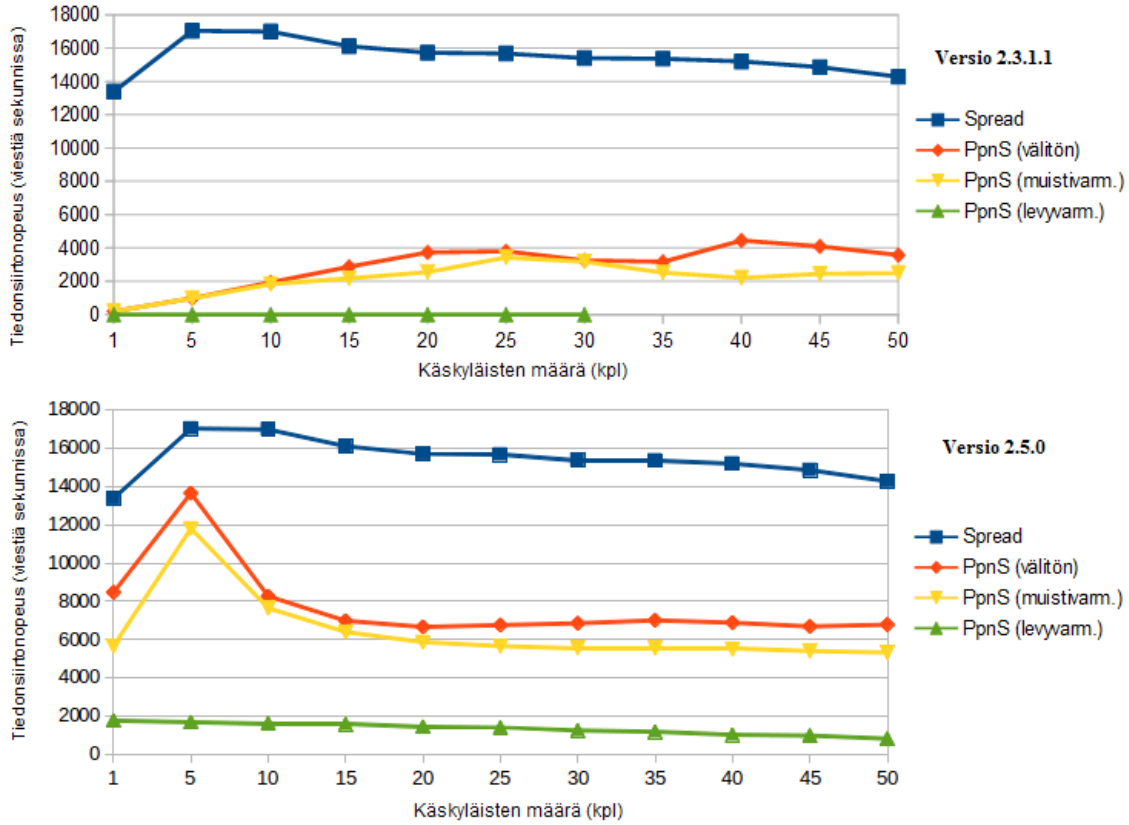
Kuvassa 30 on esitetty saman testin tulokset muodossa MB/s.



Kuva 30: Spread- ja PPnS-tasojen tiedonsiirtonopeudet viestin pituuden funktiona.

Kuvassa 30 esitetyt tuloset eivät ole täysin odotetun mukaiset. Alle 20 kt viestien nopeudet ovat loogisesti raskausjärjestyksessä. Tätä suurempien viestien kohdalla tilanne kääntyy päälaelleen ja raskaammat viestit liikkuvat nopeammin, kuin kevyet viestit. Poikkeusena on levyvarmennetut viestit, joka on aina selvästi hitain viestityyppi.

Kuvassa 31 on esitetty vastaavat mittaustulokset usean käskyläisen tapauksessa. Viestin pituus pidettiin vakiona (50 tavua), kuten alkuperäisissä mittauksissa. Ylemmässä kuvaajassa on esitetty vanhan version suorituskyky vertailun vuoksi (kuva 22).



Kuva 31: Spread- ja PPnS-tasojen tiedonsiirtonopeudet viestien pituuden funktiona usean käskyläisen tapauksessa.

Yhden käskyläisen tapauksessa tiedonsiirtonopeudet ovat käytännössä samat, kuin kuvassa 29. Kun käskyläisten määrää kasvatetaan, tiedonsiirtonopeudet lähtevät kasvuun (5 käskyläistä). PPnS-tason nopeudet kuitenkin palautuvat pian takaisin yhden käskyläisen tasolle (15 käskyläisen tapaukseen mennessä). Spread tason nopeudet sen sijaan saturoituvat vasta 10 käskyläisen kohdalla ja alkavat hiljalleen laskea. Levyvarmennettujen viestien trendi on hiljalleen laskeva käskyläisten lukumäärän kasvaessa. Tämä johtunee tietokantayhteyksien kasvavasta määrästä.

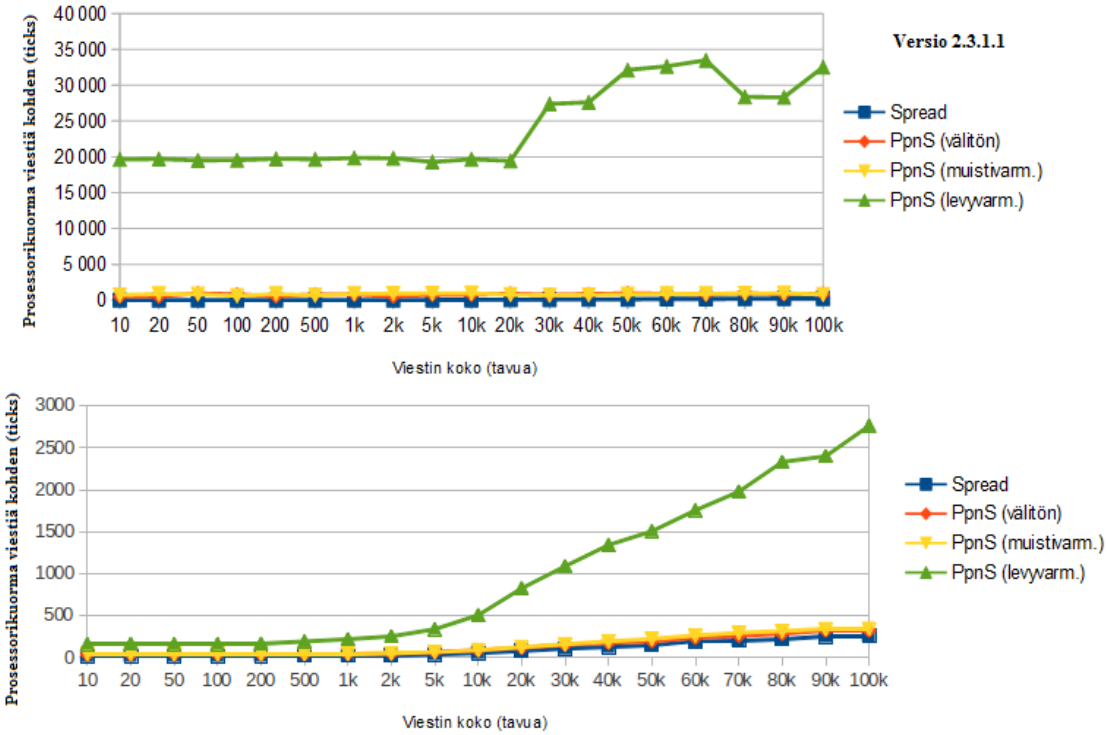
Tiedonsiirtonopeuden nousu käskyläisten lisääntyessä selittyy Spread Daemonin tavasta jakaa lähetysvuoroja useamman yhteyden tapauksessa, koska sama trendi on havaittavissa sekä Spread että PPnS -tasoilla. PPnS-tason tiedonsiirtonopeuden rajumpi lasku ei selity prosessoritehojen loppumisella, joten syy jää tämän työn puitteissa epäselväksi. Yhteenvetona voidaan todeta, että tiedonsiirtokyky parani moninkertaiseksi kaikilla PPnS:n viestityypeillä.

6.2 Prosessorikuorma

Optimointi vaikutti paljon myös prosessorikuorman määrään. Se laski kaikilla PPnS:n viestityypeillä, mutta levyvarmennetuilla viesteillä se oli aivan omaa luokkaansa. Tässä

luvussa käydään läpi luvussa 5 kuvattujen optimointien jälkeen tehdyt mittaukset ja verrataan niitä kappaleessa 4.5 esiteltyihin lähtötasomittauksiin. Kaikki prosessorikuormat mitattiin samalla tavalla, kuin lähtötasomittauksissa — niissä on mukana sekä viestien lähettämiseen että vastaanottamiseen kulunut prosessoriaika. Lisäksi niistä on poistettu testiohjelman käyttämä prosessoriaika.

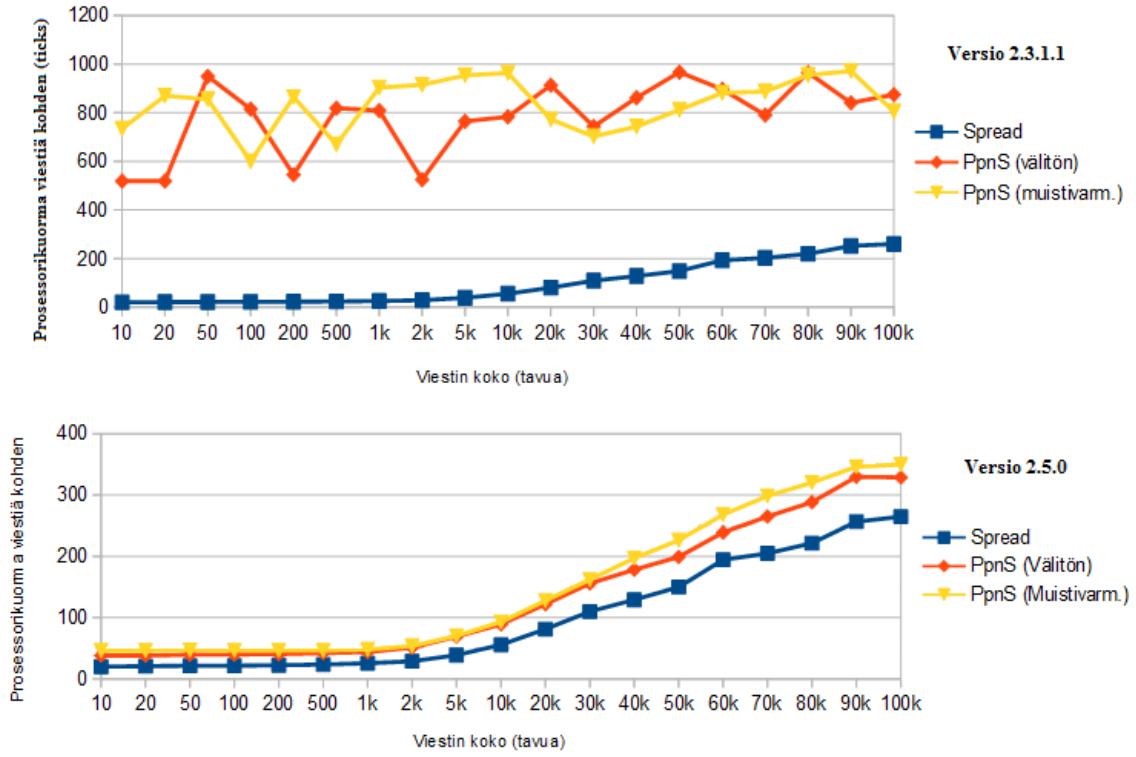
Ensin tarkastellaan viestin pituuden vaikutusta prosessorikuormaan. Käskyläisiä on aina yksi. Kuvassa 32 on esitetty prosessorikuorma toteutettujen optimointien jälkeen. Ylemmässä kuvaajassa on vanhan version tulokset vertailun helpottamiseksi.



Kuva 32: Spread- ja PPnS-tasojen prosessorikuormat viestien pituuden funktiona.

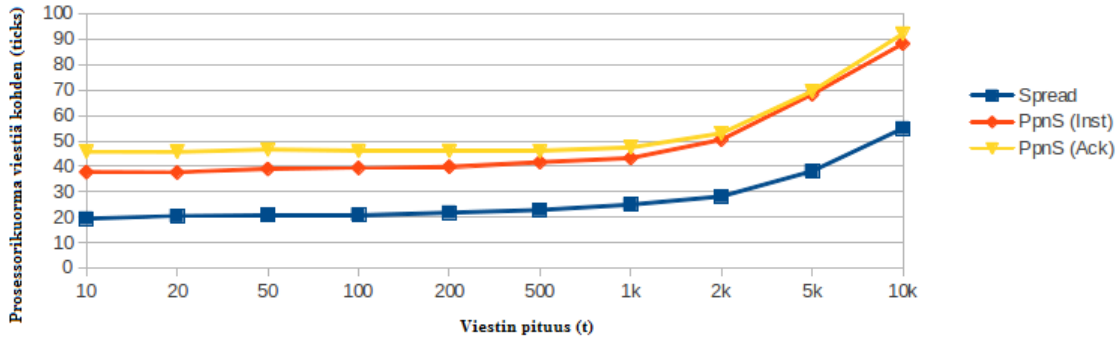
Tarkastelemalla kuvaa 32, voidaan havaita prosessorikuorman laskeneen dramaattisesti. Ennen optimointeja levyvarmennettujen viestien prosessorikuorma oli noin 20 000 – 30 000 ticks. Optimointien jälkeen vastaavat luvut olivat noin 200 – 2700. Prosessorikuorma pieneni 1/100 – 1/11.

Muiden viestityyppien prosessorikuorma on liian pieni luettavaksi yllä olevasta kuvaajasta. Kuvassa 33 on esitetty samat tulokset ilman levyvarmennettuja viestejä.



Kuva 33: Spread- ja PPnS-tasojen prosessorikuormat viestien pituuden funktiona (ilman levyvarmennettuja viestejä).

Kuvan 33 kuvaajista voidaan todeta, että optimointien jälkeen prosessorikuorma on alle kymmenesosa alkuperäisestä PPnS:n pienten välittömien ja muistivarmennettujen viestien kohdalla. Isojen viestien kohdalla prosessorikuorma ei kuitenkaan alentunut, kuin puoleen alkuperäisestä. Suurin osa viesteistä on alle kilotavun mittaisia, joten kuvassa 34 on esitetty samat mittaustulokset alle 10 kt pituisilla viesteillä.



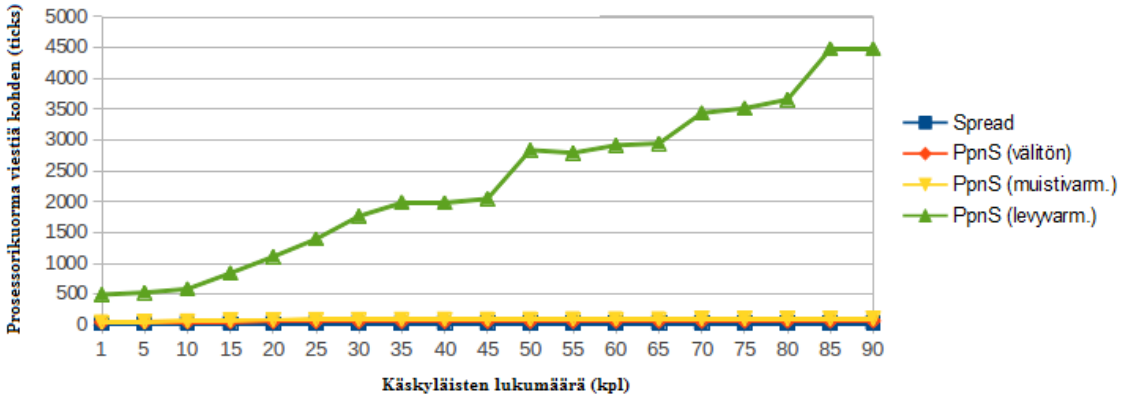
Kuva 34: Spread- ja PPnS-tasojen prosessorikuormat viestien pituuden funktiona (lyhyet viestit).

Kuvan 34 mukaan alle kilotavun viestien osalta spread-tason viestien prosessorikuorma on noin 20, välittömien 38 ja muistivarmennettujen 46. Arvoista on poistettu testiohjelman prosessorikuorma. Näiden lukujen perusteella voidaan laskea, kuinka paljon vuorontaja tuo lisäkuormaa.

Välittömien ja muistivarmennettujen viestien ainoa ero on se, että muistivarmennetut viestit menevät vuorontajan kautta, mutta välittömät lähetetään sen ohi. Voidaan olettaa, että muistivarmennettujen viestien vaatima suurempi prosessorikuorma johtuu yksistään vuorontajasta, koska viestit vastaanotetaan samalla tavalla. Vuorontajan aiheuttama prosentuaalinen prosessorikuorma koko PPnS:n käyttämästä prosessorikuormasta on

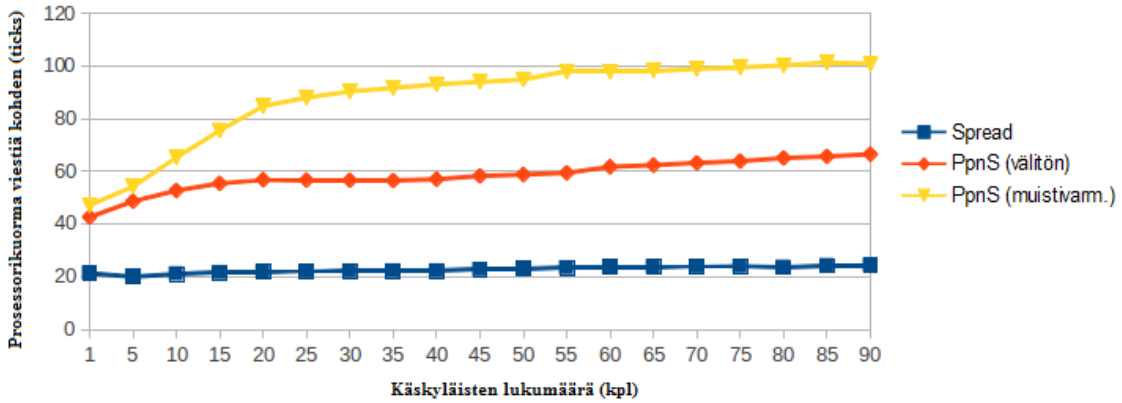
$$\frac{(ack-spread)-(inst-spread)}{ack-spread}\cdot 100\,\% = \frac{(46-20)-(38-20)}{4-20}\cdot 100\,\% \approx 30\,\%\,.$$

Seuraavaksi tarkastellaan käskyläisten määrän vaikutusta prosessorikuormaan. Kuvassa 35 on esitetty prosessorikuorma käskyläisten funktiona. Viestin pituus on vakio 50 tavua.



Kuva 35: Spread- ja PPnS-tasojen prosessorikuormat käskyläisten määrän funktiona.

Kuvan 35 arvot kuvaavat prosessorikuormaa yhtä käskyläistä kohden. Käskyläisten määrällä on selvä vaikutus levyvarmennettujen viestien prosessorikuormaan. Yhden käskyläisen käyttämä prosessoriaika on melko suoraanverrannollinen käskyläisten kokonaismäärään. Muiden viestien tulkinta ei onnistu kyseisestä kuvasta, joten samat arvot on esitetty kuvassa 36 ilman levyvarmennettuja viestejä.



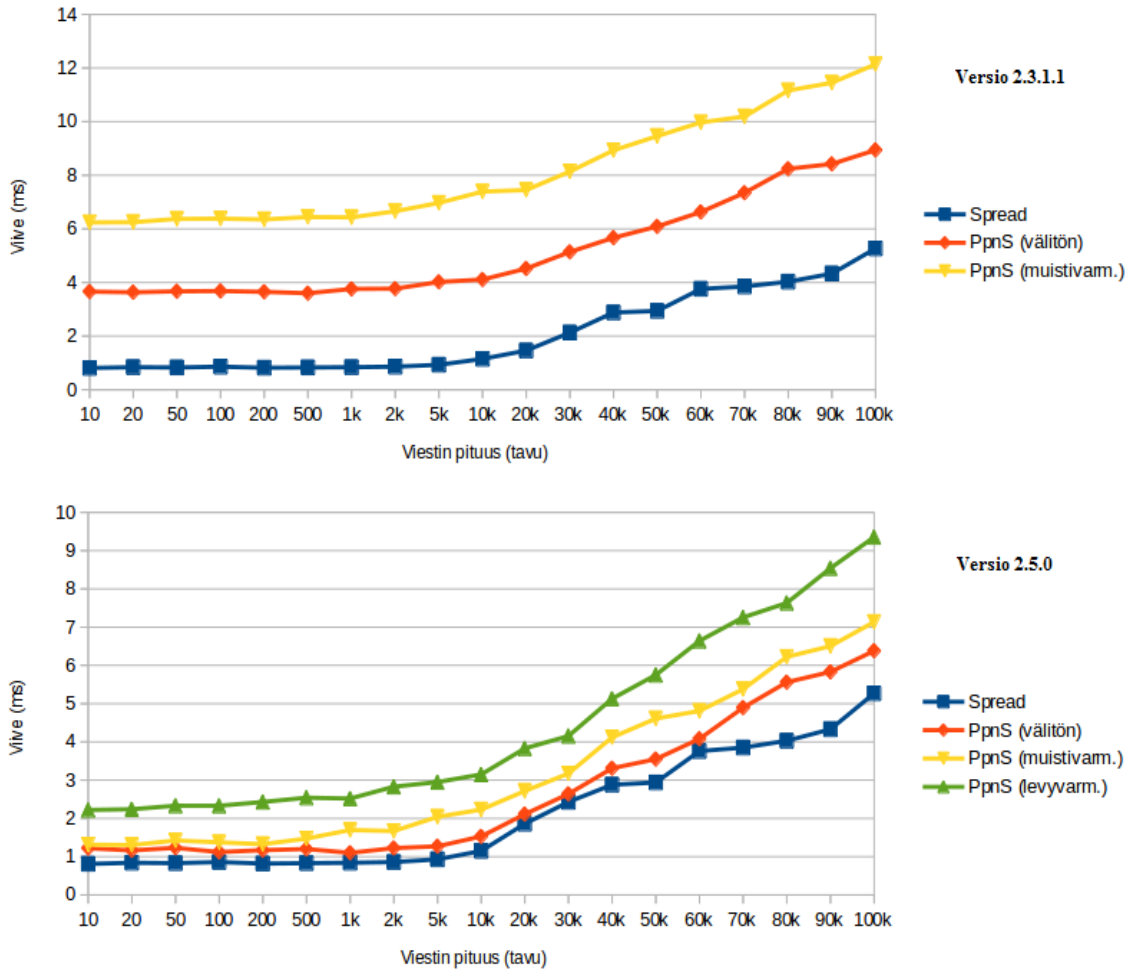
Kuva 36: Spread- ja PPnS-tasojen prosessorikuormat käskyläisten määrän funktiona (lyhyet viestit).

Kuvasta 36 näkee, että käskyläisten määrä ei vaikuta juuri ollenkaan käskyläisen käyttämään prosessoriaikaan. PPnS:n välittömien ja levyvarmennettujen viestien kohdalla prosessorikuorma kasvaa hieman käskyläisten lisääntyessä. Kuorman kasvu johtuu ainakin osittain, ellei kokonaan, oikean vastaanottaja instanssin etsimisestä. Etsimisen aikavaativuus on logaritminen, joten se selittäisi käyrien muodot.

6.3 Viive

Toteutetut optimoinnit pienensivät myös viestien välityksen viivettä. Alkuperäisessä toteutuksessa vuorontaja tarkasti lähetysjonot 5 ms:n välein. Tämä aiheutti sen, että viesti odotti jonossa turhaan keskimäärin 2,5 ms ja parhaimmillaan 5 ms. Vastaavasti lähetyspäässä viestejä käsiteltiin 5 ms:n välein, mikä aiheutti vastaavan 2,5 ms:n keskimääräisen viiveen. Yhteensä turhaa jonoissa odottelua syntyi varmennetuilla- ja levyvarmennetuilla viesteillä keskimäärin 5 ms ja parhaimmillaan 10 ms. Välittömillä viestellä turhaa odottelua tapahtui vain vastaanottajan päässä, joten keskimääräinen jonossa odottelu jäi 2,5 ms:n.

Tässä luvussa käydään läpi luvussa 5 kuvattujen optimointien jälkeen tehdyt viivemittaukset ja verrataan niitä kappaleessa 4.5 esiteltyihin lähtötasomittauksiin. Kuva 37 esittää viivettä viestin pituuden funktiona yhden käskyläisen tapauksessa. Ylempi kuvaaja esittä vanhan version tuloksia.



Kuva 37: Viestin välityksen viive viestin pituuden funktiona.

Vertailemalla kuvan 37 kuvaajia, voidaan todeta viiveen pienentyneen selvästi kaikkien PPnS:n viestityyppien kohdalla. Ylemmässä kuvaajassa ei ole levyvarmennettujen viestien viiveen käyrää, mutta sen arvo on noin 140 ms riippumatta viestin pituudesta (kts. kuva 26). Ylivoimaisesti eniten pieneni levyvarmennettujen viestien välittämisen viive. Viive pieneni noin 16 – 55 kertaa pienemmäksi, kuin alkuperäinen. Viiveen pienentyminen johtui lähes kokonaan tietokannan optimoinnin osalta, mutta muutamia millisekunteja lähti myös vuorontajan, sekä viestien vastaanottamisen optimoinnin seurauksena. Muistivarmennettujen viestien kohdalla viive pieneni teoreettiset 5 ms, joten voidaan päätellä sen johtuneen pääasiassa vuorontajan ja viestien vastaanottamisen optimoinneista. Välittömien viestien kohdalla suurin aikuttava tekijä lienee viestien vastaanottamisen optimointi, koska viive pieneni teoreettiset 2,5 ms.

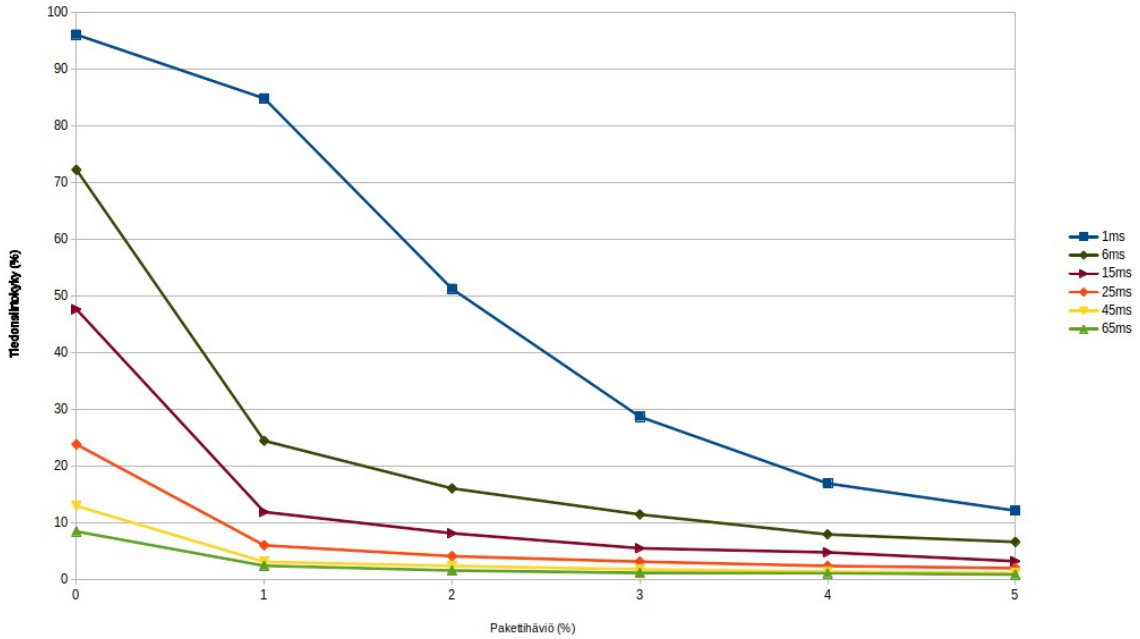
6.4 TCP pullonkaulana

TCP on suuri pullonkaula huonoisssa langattomissa verkoissa. Sen käyttö on kuitenkin välttämätöntä, jotta verkon kapasiteetti ei pääse ylikuormittumaan.

Varsinaisten suorituskykymittausten ohessa tutkittiin PPnS:n tiedonsiirtokyvyn käyttäytymistä huonoissa verkoissa, joissa viive ja pakettihäviöt ovat merkittäviä. Tiedonsiirtokykyä mitattiin kappaleessa 4.2.1 kuvatulla menetelmällä. Erona oli se, että viestit reititettiin WANem:n kautta, mikä mahdollisti viiveen ja pakettihäviön simuloinnin. Lähetettävien viestien koko oli 5 kt ja verkon Mittauksiin valittiin kaksi tunnettua ruuhkanhallintaikkuna-algoritmia (eng. Congestion control window algorithm): TCP Cubic ja TCP Westwood.

TCP Cubic on Linux-kernelien oletusruuhkanhallintaikkuna-algoritmi, joka on suunniteltu suurinopeuksiin verkkoihin, joissa on paljon viivettä [6]. Cubic on täten luonnollinen valinta vertailun pohjaksi. TCP Westwood on sitä vastoin suunniteltu toimimaan paremmin verkoissa, joissa esiintyy jonkin verran viivettä ja pakettihäviötä [15].

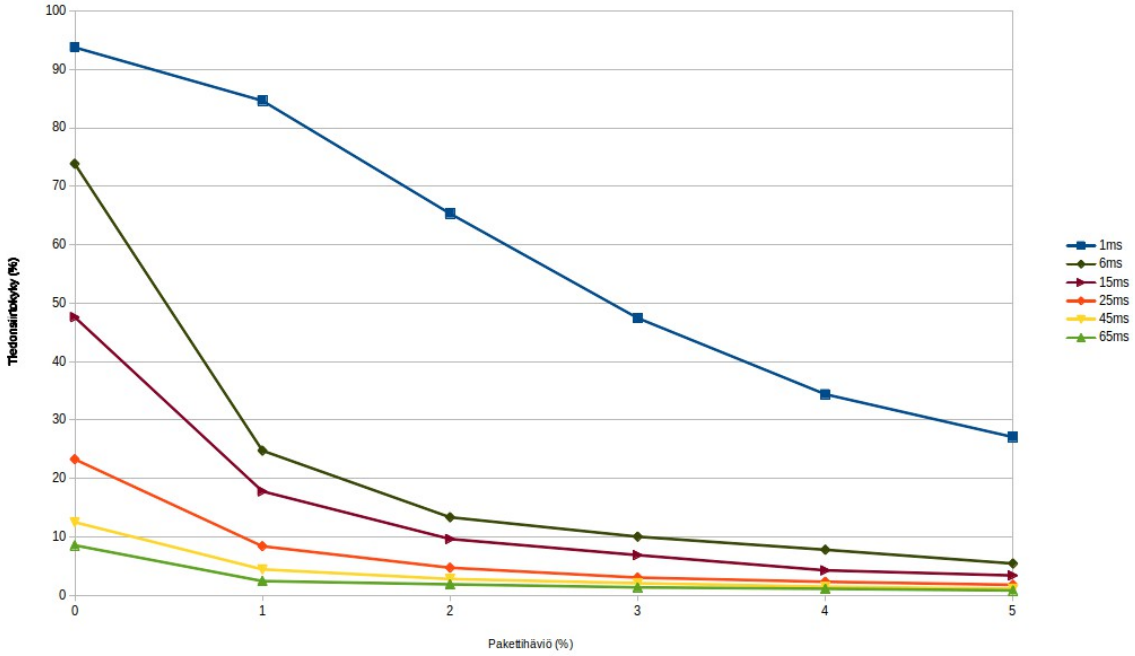
Kuva 38 esittää PPnS:n tiedonsiirtokykyä muistivarmennettujen viestien osalta TCP Cubic:n tapauksessa, kun verkon viive ja pakettihäviö kasvavat.



Kuva 38: Viiveen ja pakettihäviön vaikutus PPnS:n tiedonsiirtokykyyn TCP Cubic -ruuhkanhallintaikkuna-algoritmin tapauksessa.

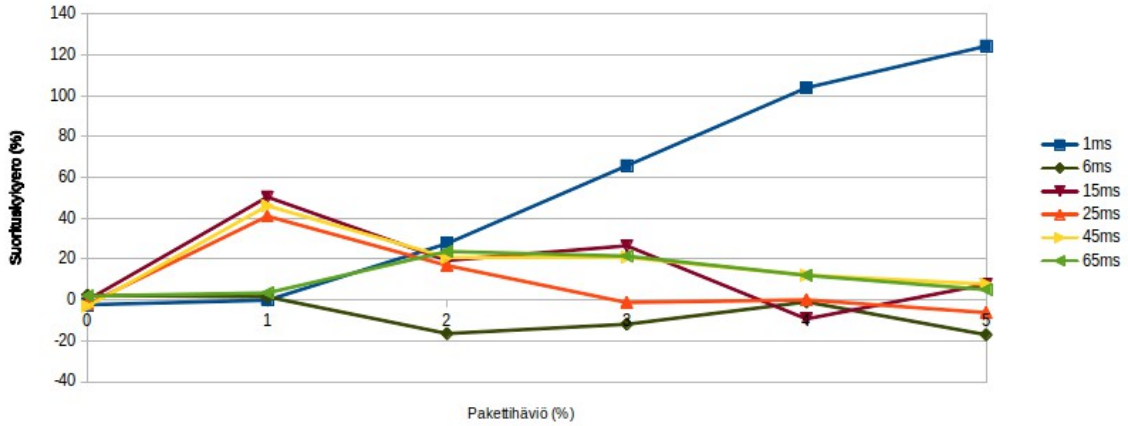
Kuvasta 38 näkee, että PPnS:n tiedonsiirtokyky käyttäytyy, kuten TCP:n tiedonsiirtokyky viiveen ja pakettihäviön kasvaessa. Kun viive pysyy maltillisena, pienellä pakettihäviöllä ei ole järisyttävää vaikutusta. Viiveen kasvaessa tiedonsiirtokyky putoaa dramaattisesti.

Tiedonsiirtokykyä voidaan parantaa käyttämällä sopivaa ruuhkanhallintaikkuna-algoritmia. Kuva 39 esittää vastaavat mittaustulokset TCP Westwood:n tapauksessa.



Kuva 39: Viiveen ja pakettihäviön vaikutus PPnS:n tiedonsiirtokykyyn TCP Westwood -ruuhkanhallintaikkunan tapauksessa.

Kuvasta 39 näkee, että tulokset ovat melko saman suuntaisia, kuin TCP Cubic:n tapauksessa. Kasvava viive vaikuttaa dramaattisesti tiedonsiirtokykyyn ja pakettihäviöllä on myös selvä vaikutus. Erot eivät ole niin dramaattisia, että ne näkyisivät selvästi vertailemalla kuvia 38 ja 39, joten kuva 40 esittää, kuinka monta prosenttia nopeammin Westwood siirtää tietoa Cubiciin nähden.



Kuva 40: TCP Cubicin tiedonsiirtokyvyn ero TCP Westwoodin tiedonsiirtokykyyn nähden.

Westwood pärjää lähes järjestäen paremmin, kuin Cubic. Tämä näkyy erityisesti silloin, kun viive tai pakettihäviö ovat pieniä. Viiveen ollessa pieni (1 ms), Westwood on

suhteessa parempi, mitä enemmän pakettihäviötä esiintyy. Pakettihäviön ollessa 5 %, tiedonsiirtokyky on jopa 120 % parempi Cubiciin verrattuna. Vastaavasti pakettihäviön ollessa pieni (1 %), viiveen vaikutus ei ole niin radikaali (viiveiden 15, 25 ja 45 käyrät) — Westwood:n tiedonsiirtokyky on 40 % parempi, kuin Cubic:n.

Westwood ei hallitse kuitenkaan kaikkia tilanteita Cubicia paremmin. Tarkastelemalla 6 ms:n käyrää, voidaan havaita Cubic pärjäävän kaikilla mitatuilla pakettihäviömäärillä vähintään yhtä hyvin, kuin Westwood. 6 ms on hyvin tyypillinen viive todellisissa verkoissa, joten Westwoodin käyttö Cubicin sijasta ei välttämättä tuo käytännön hyötyä. Pahimmassa tapauksessa siitä on vain haittaa.

6.5 Tulosten tarkkuus ja virhelähteet

Mittaustarkkuus on aina rajallinen ja virhelähteet on syytä ottaa huomioon tuloksia analysoitaessa. Tässä diplomityössä suorituskykymittausten tuloksien tarkkuudeksi riitti suurpiirteinen tarkkuus. Tämä siksi, että alustaa ajetaan useilla eri laitteilla, joiden laskentatehot vaihtelevat paljon. Oleellista oli selvittää, kuinka suuriin tiedonsiirtonopeuksiin alusta kykenee. Tarkkuudeksi riitti, että alle kilotavun mittaisia viestejä pystytään välittämään tyypillisen pöytäkoneen tehoisella laitteella 6000 kpl/s.

Optimoimattoman ja optimoidun version välisessä vertailussa oleellista oli osoittaa, että toteutetulla optimoinnilla oli käytännön vaikutus. Tässä oleellista oli pitää muu ympäristö samana ja muuttaa ainoastaan optimoitavaa ohjelmaa. Seuraavaksi käydään läpi huomionarvoiset virhelähteet ja muut tuloksiin vaikuttavat tekijät sekä tavat, joilla niitä pyrittiin minimoimaan.

Suurin tuloksiin vaikuttava tekijä on laitteen laskentateho. Se vaikuttaa kaikkien viestityyppien kohdalla sekä tiedonsiirtonopeuteen että viestin välityksen viiveeseen. Muistivarmennettujen viestien kohdalla merkittävä tekijä on kiintolevyn nopeus. Se vaikuttaa vastaavasti sekä tiedonsiirtonopeuteen että viestin välityksen viiveeseen. Kolmas vaikuttava tekijä on verkkolaitteet. Verkkolaitteilla tarkoitetaan verkkokortteja, -kytkintä ja -johtoja. Yksikin heikkolaatuinen osa voi laskea linkin nopeutta merkittävästi. Yhdelläkään edellä mainituista tekijöistä ei kuitenkaan ole vaikutusta vertailtaessa optimoimattoman ja optimoidun version suorituskykyjen eroja.

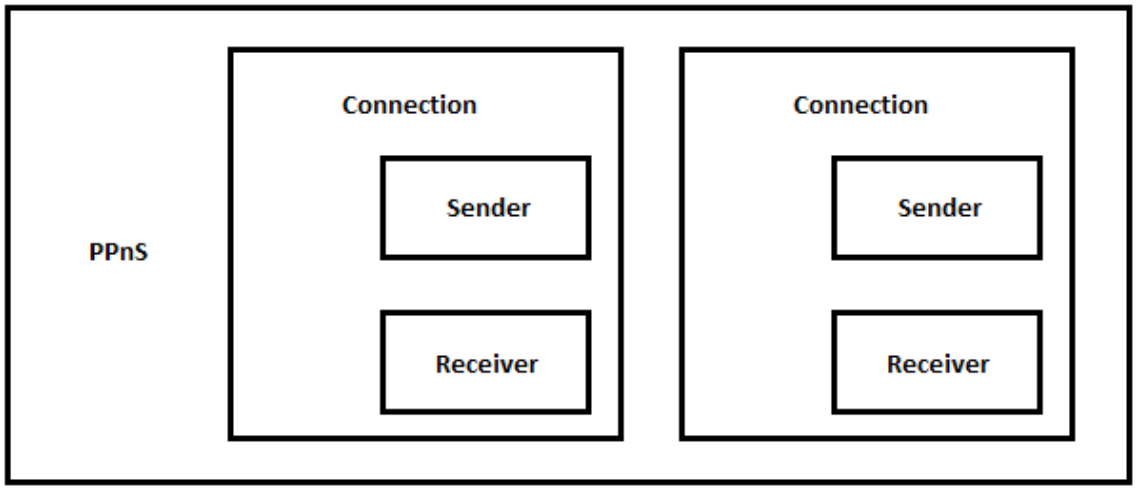
Vertailumittauksissa merkittäviä virhelähteitä olivat lähinnä satunnaiset häiriöt. Esimerkkeinä tällaisista ovat jonkin taustaprosessin aiheuttama hetkellinen tai pitempiaikainen piikki prosessorikuormassa tai kiintolevyn käytössä. Tämän tyyppiset virhelähteet pyrittiin minimoimaan sulkemalla mittauksien ajaksi muut ohjelmat sekä ajamalla mittauksia useita kertoja ja laskemalla niistä keskiarvo.

7 JATKOKEHITYSIDEAT

Kuten edellä on mainittu, tämän diplomityön puitteissa ei toteutettu kaikkia mieleen tulleita optimointeja, vaan osa jätettiin odottamaan myöhempää ajankohtaa. Suurimpia syitä olivat ajan puute ja se, ettei kriittiseen ohjelman osaan haluttu tehdä suuria muutoksia, jotka mahdollisesti aiheuttaisivat muita ongelmia. Tässä kappaleessa on esitelty keskeisimmät jatkokehitysideat, joilla olisi vaikutusta tiedonsiirtonopeuteen tai prosessorikuormaan.

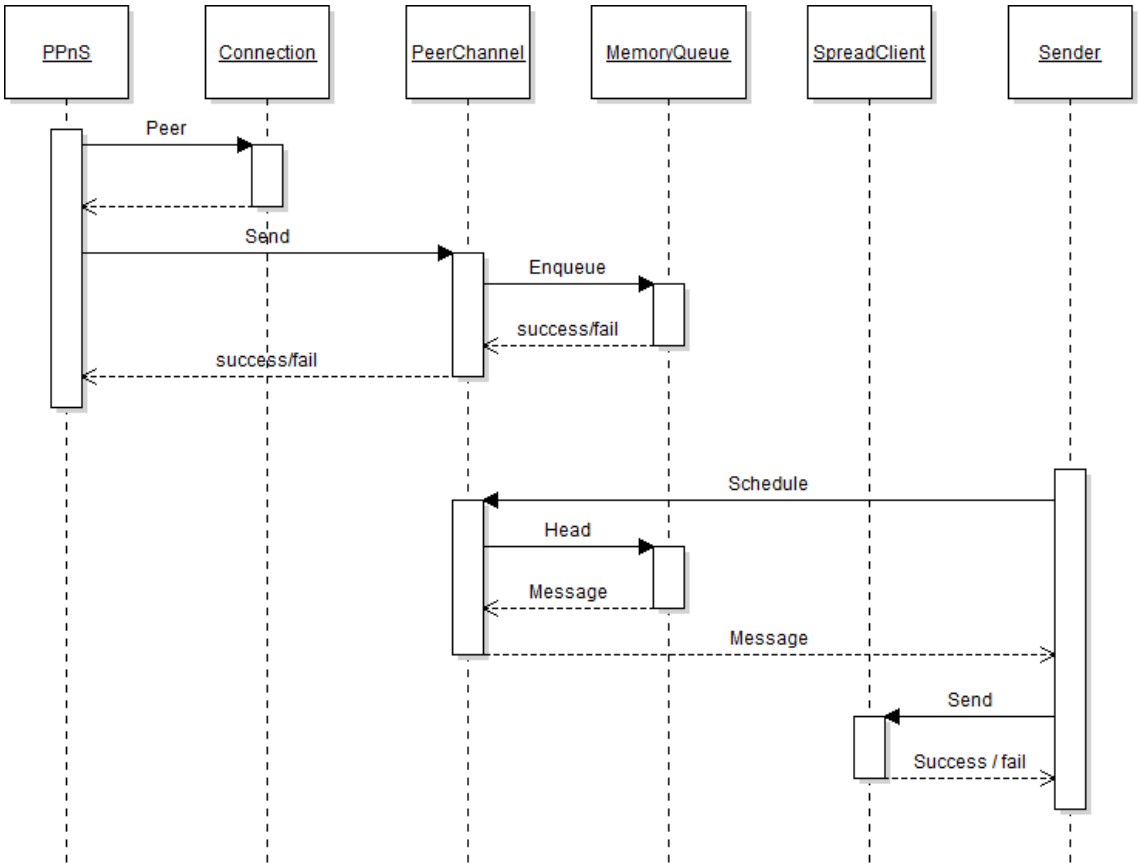
7.1 Viestien lähettäminen

Tapa, jolla viestit lähetetään lopullisessa versiossa on kuvattu kappaleessa 5.2. Toteutettu tapa on helppo ja yksinkertainen muutos alkuperäiseen toteutukseen. Se on kuitenkin hyvin kömpelön oloinen ratkaisu ja se tulisi korvata paremmalla. Yksi vaihtoehto olisi käyttää säikeitä tuottaja-kuluttaja -mallin mukaisesti. Kuva 41 esittää kaaviota PPnS:n tarvittavasta säikeistyksestä.



Kuva 41: Kaaviokuva tarvittavasta säikeistyksestä.

PPnS:n säikeistys on nykyisin lähes samanlainen. Itse PPnS-luokka toimii pääsäikeessä. Jokaisella yhteydelle (Connection) luodaan oma säije ja jokaisessa Connection-säikeessä on oma säije viestien vastaanottamiselle (Receiver). Uudessa mallissa lisättäisiin Connection-säikeeseen Receiver-säikeen rinnalle uusi säije lähettämistä varten (Sender). Sender-säikeen tehtävänä olisi odottaa semaforin takana uusia viestejä ja huolehtia niiden lähettämisestä. Kuva 42 esittää ajoituskaaviota ehdotetusta toteutuksesta.



Kuva 42: Ehdotetun vuorontajan ajoituskaavio.

Viesti kirjoitetaan jonoon, kuten nykyisessä toteutuksessa. PPnS pyytää Connection luokalta *Peer*-metodilta oikean PeerChannel-instanssin, jonka jälkeen kutsutaan saadun instannsin *Send*-metodia. PeerChannel puolestaan kutsuu MemoryQueuen *Enqueue*-metodia lisätäkseen viestin, jonka jälkeen viesti on jonossa odottamassa vuorontajaa. Tämä kaikki tapahtuu Connection-säikeessä (kts. kuva 41).

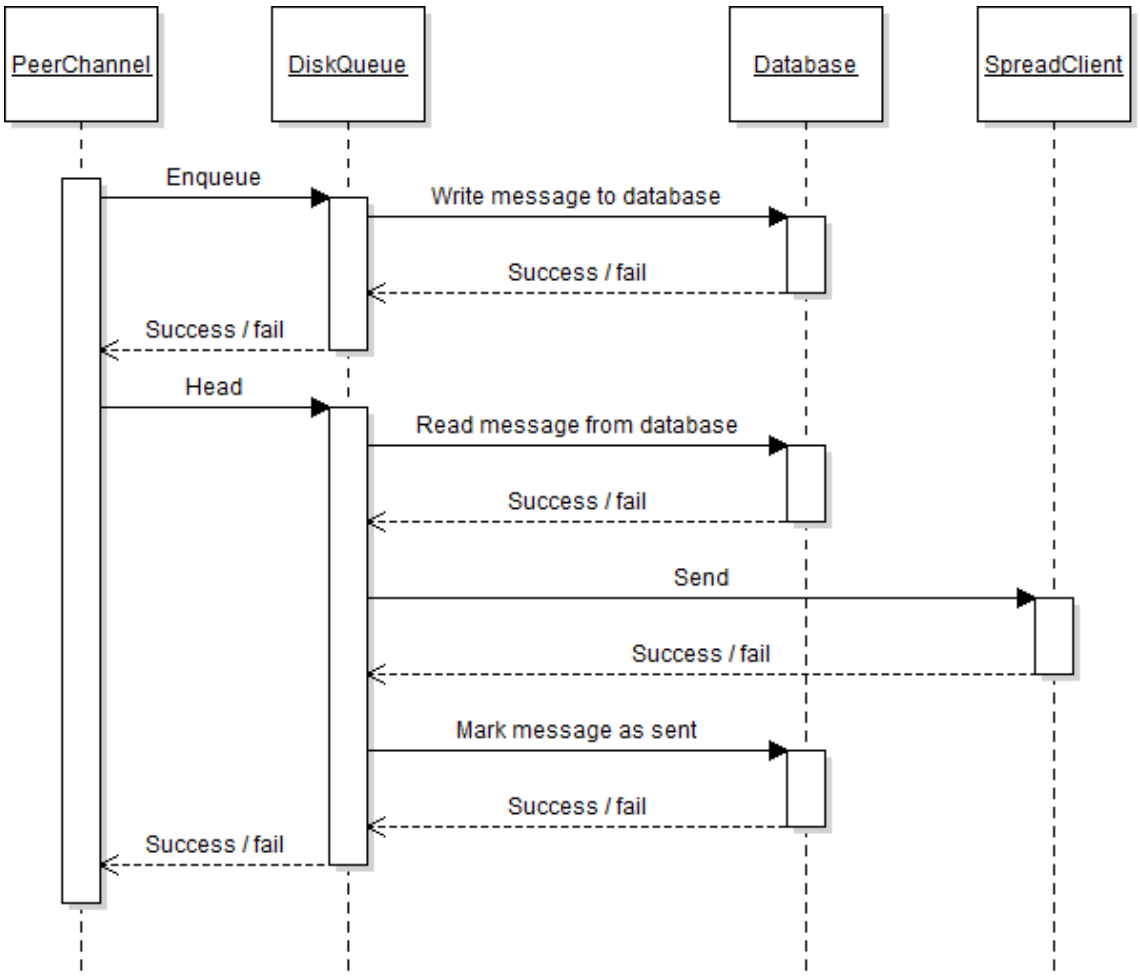
Kuvan alempi sekvenssi suoritetaan erillisessä *Sender*-säikeessä. Sender-luokka kutsuu PeerChannel-luokan Schedule-metodia, joka pyytää MemoryQueue-luokalta seuraavan lähetettävän viestin Head-metodilla. Viesti välitetään paluuarvoina aina Sender-luokalle asti, joka kutsuu SpreadClient-luokan Send-metodia viestin lähettämiseksi.

Viestipuskuri suojataan semaforilla (eng. semaphore). Semaforin arvo on sama, kuin viestipuskurin yhteenlaskettu viestimäärä. Se alustetaan aluksi nollaksi. Kun jonoon lisätään viesti, semaforin arvoa kasvatetaan yhdellä. Kun jonosta poistetaan viesti, semaforin arvoa pienennetään yhdellä. Semaforin arvoa kasvatetaan myös silloin, kun viestejä palautetaan uudelleenlähetettäväksi.

7.2 Tietokanta

Muistivarmennettujen viestien lähetyksessä suurin pullonkaula on tietokantakutsujen nopeus. Nopeutta pystytään lisäämään huomattavasti, jos kutsujen määrää voidaan vähentää, joko poistamalla turhia tai yhdistelemällä useampia kutsuja yhdeksi kutsuksi.

Nykyisessä toteutuksessa on mahdollista tehdä molempia. Muistivarmennettujen viestien lähettäminen on kuvattu aiemmin kuvassa 11. Kuvassa 43 on ajoituskaavio, joka kuvaa niiden lähettämisen yksittäisten tietokantakutsujen osalta.

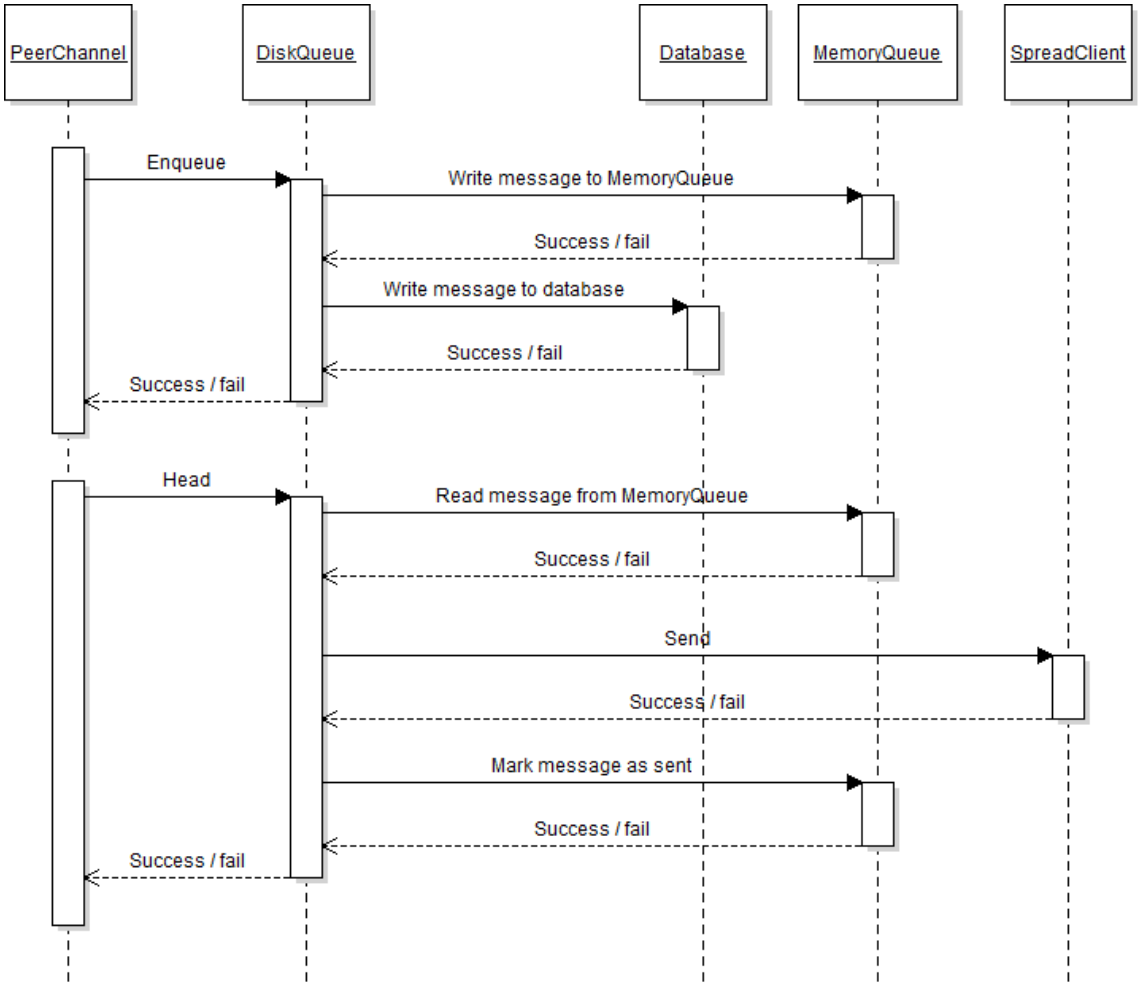


Kuva 43: Muistivarmennettujen viestien lähettäminen tietokantakutsujen osalta.

Viestin lisääminen jonoon on suoraviivainen operaatio tietokantakutsujen osalta. Kantaan lisätään yksi rivi viestitauluun. Viestitaulussa on viestin järjestysnumero, viestin sisältö sekä tieto siitä, onko viesti lähetetty. Viestin lukeminen ja lähettäminen vaatii kaksi erillistä tietokantakutsua. Ensin viesti luetaan kannasta, jonka jälkeen se yritetään lähettää. Jos viestin lähettäminen onnistuu, viesti merkataan lähetetyksi tietokantaan.

Nykyisessä toteutuksessa kaikki viestit kirjoitetaan ensin kantaan, jonka jälkeen ne luetaan sieltä ja lähetetään. Viestejä ei ole pakko lukea kannasta ennen niiden lähetystä,

koska rinnalla voidaan ylläpitää muistijonoa, josta viestin lukeminen on paljon nopeampaa. Tämä ei kuitenkaan poista tarvetta merkata viestejä lähetetyiksi. Kuva 44 esittää ehdotetun toteutuksen ajoituskaaviota.



Kuva 44: Ehdotetun toteutuksen ajoituskaavio tietokantakutsujen osalta.

Kun viesti lisätään jonoon, se kirjoitetaan tietokannan (DiskQueue) lisäksi muistissa olevaan jonoon (MemoryQueue). Lähetettäessä viestin sisältö luetaan muistista tietokannan sijaan. Lähetyksen jälkeen viesti pitää vielä merkata lähetetyksi. Lähetetyksi merkkaamista voidaan nopeuttaa yhdistämällä useamman peräkkäisen viesti merkkaaminen yhdeksi kutsuksi. Tämä siksi, että joskus tapahtuva muutaman viestin turha uudelleen lähetys ei ole vaarallista.

8 YHTEENVETO

Tämän diplomityön tavoitteena oli analysoida UniQ-alustan suorituskykyä ja pyrkiä optimoimaan sitä. Analysoinnin tavoitteena oli pyrkiä selvittämään tiedonsiirtoalustan pullonkaulat sekä saamaan kuva sen suorituskyvystä. Toisena tavoitteena oli optimoida tiedonsiirtoalustaa viestien välityksen, viiveen ja prosessorin käytön osalta.

Ongelmia lähdettiin ratkaisemaan mainitussa järjestyksessä. Ensin toteutettiin testiohjelma, joilla mitattiin alustan tiedonsiirtokerrosten ja viestityyppien suorituskykyä. Mittaustulokset dokumentoitiin ja todettiin PPnS-tason olevan suurin pullonkaula tiedonsiirrossa. Tämän jälkeen optimoinnin kohteita etsittiin Walgrind ohjelman ja koodikatselmoinnin avulla. Löydetyt pullonkaulat optimointiin ja niiden vaikutus suorituskykyyn todennettiin edellä mainitulla mittausohjelmalla. Optimointien toteutusta rajoitti projektin vastuuhenkilöiden vastahakoisuus suurempien muutosten toteuttamiseen. Tästä syystä osa optimoinneista päätettiin toteuttaa myöhemmin. Lopuksi mitattiin ja dokumentoitiin kaikkien toteutettujen optimointien yhteisvaikutus.

Työn tuloksena alustan suorituskyky parani merkittävästi lähes kaikilla analysoiduilla osa-alueilla. Suurin käytännön hyöty saavutettiin viestien lähetysnopeudessa sekä prosessorikuormassa. Viestien viive pieneni paljon prosentuaalisesti, mutta käytännön merkitys jäänee vähäiseksi muutamaa poikkeusta lukuun ottamatta. Taulukkoon 3 on koottu yhteenveto saavutetuista tuloksista. Luvut tarkoittavat, kuinka moninkertainen parannus oli alkuperäiseen nähden. Tiedonsiirtokyvyn kohdalla tämä tarkoittaa välityskyvyn kasvua, mutta prosessorikuorman ja viiveen kohdalla niiden pienenemistä. Suluissa olevat luvut tarkoittavat lopullista arvoa. Kaikki luvut ovat karkeita kuvaajista luettuja likiarvoja.

Taulukko 3: Yhteenveto saavutetuista suorituskykyparannuksista

Suorituskyvyn tyyppi	**Viestityyppi**	**Lyhyet viestit (< 200 tavua)**	**Pitkät viestit (n. 100 kt)**
Tiedonsiirtokyky	Välitön	40 (8000 viestiä/s)	2 (400 viestiä/s)
	Muistivarmennettu	30 (6000 viestiä/s)	2 (400 viestiä/s)
	Levyvarmennettu	200 (1900 viestiä/s)	4 (20 viestiä/s)
Prosessorikuorma	Välitön	15	5
	Muistivarmennettu	15	5
	Levyvarmennettu	65	10
Viive	Välitön	3 (1,5 ms)	1,5 (6 ms)
	Muistivarmennettu	4 (1,5 ms)	2 (7 ms)
	Levyvarmennettu	55 (2,5 ms)	15 (9 ms)

Tiedonsiirtokyky parani kaikkien viestityyppien osalta monikymmenkertaiseksi. Levyvarmennettujen viestien kohdalla jopa 200 kertaiseksi. Näin suuri parannus oli mahdollista, koska viestien lähetyksen vuoronnus oli toteutettu suorituskykyä ajattelematta. Viestejä lähetettiin yksi kerrallaan 5 ms:n välein. Viestejä myös vastaanotettiin yksi kerrallaan 5 ms:n välein. Lisäksi levyvarmennettujen viestien kohdalla tietokannan suorituskykyä ei oltu mietitty ollenkaan.

Prosessorikuorma pieneni myös kertaluokan verran kaikilla viestityypeillä. Parannukset johtuivat pääasiassa samoista syistä, kuin tiedonsiirtokyvyn kasvu. Lisäksi viestien vastaanottaminen oli toteutettu raskaalla tavalla.

Viestien välityksen viive pieneni murto-osaan välittömien ja muistivarmennettujen viestien osalla. Levyvarmennettujen viestien kohdalla muutos oli jopa monikymmenkertainen. Nämä parannukset johtuivat pääasiassa samoista syistä, kuin tiedonsiirtokyvyn paraneminen.

Saavutettujen suorituskykyparannusten lisäksi suorituskykymittaukset toimivat hyvinä rasitustesteinä tiedonsiirtoalustalle. Mittausten aikana havaittiin useita kaatumisiin ja lukkiutumisiin (eng. deadlock) johtavia virheitä, mitkä saatiin paikannettua ja korjattua testien avulla.

LÄHTEET

[1] The Spread Toolkit,
saatavilla: http://www.spread.org/index.html, vierailtu: marraskuu 2016.

[2] Wikipedia: IRC,
saatavilla: https://fi.wikipedia.org/wiki/IRC, vierailtu: lokakuu 2016.

[3] William Stallings: High speed networks and internet, performance and quality of service, second edition.

[4] Claudio Casetti, Mario Gerla, Saverio Mascolo, M.Y. Sanadidi and Ren Wang. Injong Rhee and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant, saatavilla: https://pdfs.semanticscholar.org/2d49/06884bc5309f1539195ff5b181d41a15ff60.pdf, vierailtu: marraskuu 2016.

[5] TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks
saatavilla: http://nets.cs.ucla.edu/publication/download/41/Wn_02.pdf,
vierailtu: marraskuu 2016.

[6] Wikipedia: Cubic TCP,
saatavilla: https://en.wikipedia.org/wiki/CUBIC_TCP, vierailtu: syyskuu 2016.

[7] WANem: Home
saatavilla: http://wanem.sourceforge.net/, vierailtu: elokuu 2016.

[8] WANem: Screenshots
saatavilla: http://wanem.sourceforge.net/screen.html, vierailtu: elokuu 2016.

[9] Valgrind,
saatavilla: http://valgrind.org/, vierailtu: lokakuu 2016.

[10] Wikipedia: SQLite,
saatavilla: https://fi.wikipedia.org/wiki/SQLite, vierailtu: maaliskuu 2016.

[11] SQLite documentation: Pragma statements,
saatavilla: http://www.sqlite.org/pragma.html, vierailtu: maaliskuu 2016.

[12] SQLite documentation: SQLite's Use of Temporary Disk Files,
saatavilla: http://www.sqlite.org/tempfiles.html, vierailtu: huhtikuu 2016.

[13] SQLite documentation: Write-ahead logging,
saatavilla: http://www.sqlite.org/wal.html, vierailtu: huhtikuu 2016.

[14] Wikipedia: Query Plan,
saatavilla: https://en.wikipedia.org/wiki/Query_plan, vierailtu: maaliskuu 2016.

[15] Wikipedia: TCP Westwood,
saatavilla: https://en.wikipedia.org/wiki/TCP_Westwood,
vierailtu: syyskuu 2016.